NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

AN ENHANCED GRAPHICAL USER INTERFACE FOR ANALYZING THE VULNERABILITY OF ELECTRICAL POWER SYSTEMS TO TERRORIST ATTACKS

by

Dimitrios Stathakos

December 2003

Thesis Advisor:	Javier Salmeron
Second Reader:	Kevin Wood

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: An Enhanced Graphical User Interface For Analyzing The Vulnerability Of Electrical Power Systems To Terrorist Attacks.	**5. FUNDING NUMBERS**
6. AUTHOR(S) Dimitrios Stathakos	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis develops a Graphical User Interface (GUI) to represent electric power grids subject to interdiction (attack) by terrorists. The work enhances the prototypic One-line Diagram (OD) representations of electric power networks in the VEGA 1.0 decision-support system (Vulnerability of Electrical Power Grids Analysis, version 1.0). Conforming to Windows standards, the new OD GUI incorporates advanced graphical features, which help the user visualize the model and understand the consequences of interdiction. The new ODs also capture the details of system restoration over time following an attack. The enhanced OD GUI has been incorporated into the updated version of the system, VEGA 2.0.

14. SUBJECT TERMS Graphical User Interface, Electrical Power Systems, Visual Basic			**15. NUMBER OF PAGES** 123
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

**AN ENHANCED GRAPHICAL USER INTERFACE FOR
ANALYZING THE VULNERABILITY OF ELECTRICAL POWER SYSTEMS TO
TERRORIST ATTACKS**

Dimitrios A. Stathakos
Major, Hellenic Army
Hellenic Army Military Academy, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

**NAVAL POSTGRADUATE SCHOOL
December 2003**

Author: Dimitrios Stathakos

Approved by: Javier Salmeron
                 Thesis Advisor

                 Kevin Wood
                 Second Reader

                 Jim Eagle
                 Chairman, Department of Operations Research

iii

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis develops a Graphical User Interface (GUI) to represent electric power grids subject to interdiction (attack) by terrorists. The work enhances the prototypic One-line Diagram (OD) representations of electric power networks in the VEGA 1.0 decision-support system (Vulnerability of Electrical Power Grids Analysis, version 1.0). Conforming to Windows standards, the new OD GUI incorporates advanced graphical features, which help the user visualize the model and understand the consequences of interdiction. The new ODs also capture the details of system restoration over time following an attack. The enhanced OD GUI has been incorporated into the updated version of the system, VEGA 2.0.

THIS PAGE INTENTIONALLY LEFT BLANK

DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made to ensure that the programs are free of computational and logic errors, they cannot be considered fully validated. Any application of these programs without the additional verification is at the risk of the planner.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Sincere Appreciation to

Prof Javier Salmeron   &   Prof Kevin Wood

For their patience and support.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis develops an improved graphical user interface (GUI) for the analysis tool called VEGA (Vulnerability of Electrical Power Grids Analysis). VEGA represents electric power grids subject to interdiction (attack) by terrorists and helps to detect vulnerabilities and to identify system upgrades that will harden the grid against attack (e.g., security-system improvements, capacity expansion for backup purposes, etc.). This work contributes to the research project "Optimizing the Electric Grid Design Under Asymmetric Threat." The need for analysis stems from concern about the potentially catastrophic consequences to the United States' economy and security that would result from a well-planned terrorist attack on its electric power grid.

The VEGA system comprises a GUI, a supporting database (DB) and optimization tools. The GUI helps prepare power network data for analysis, and then displays analytical results, by enabling easy navigation through customized tables and graphics containing problem data and results. This gives a user easy access to the mathematical analysis of a problem even if he or she is not an expert in mathematical modeling and optimization.

The thesis enhances an instrumental component of the VEGA GUI called the One-line Diagram (OD) GUI. ODs are often used by electrical engineers to represent electric power networks. The enhanced OD GUI incorporates advanced features through a number of dynamic graphics that help the user visualize the model and understand the effects of interdiction. These effects include the quantity and location of "load shedding" (unmet demand for electrical energy), and how this load shedding changes as the grid is repaired after an attack.

The VEGA optimization module performs the mathematical analysis of the problem independently of the GUI. The purpose of the GUI is to prepare the case data to be analyzed and to retrieve and display the optimization results in a

user-friendly fashion. The GUI in the existing system prototype, VEGA 1.0, serves the first purpose well, and results can also be visualized in tabular form. However, the OD representation in Vega 1.0 is limited and lacks the flexibility required by DoD Human-Computer-Interface standards. Examples of needed enhancements include object mobility and resizing, use of dialogues, zooming, printing, and more extensive use of menus, controls, toolbars, and secondary windows. The OD GUI developed in this thesis provides these enhancements.

In order to overcome certain graphical-design limitations in the Visual Basic (VB) programming language in which the VEGA GUI is implemented, we have incorporated a commercial ActiveX control called AddFlow. AddFlow is a flexible programming object that allows us to create generic network diagrams. We have customized the AddFlow object to generate ODs satisfying our representational needs. This has been accomplished by adapting AddFlow's own features and combining them with other VB features.

The resulting GUI and its supporting relational DB are crucial for organizing planning data, reducing clerical error through embedded validations, completing missing details, filtering information according to user's needs, and displaying multiple scenarios with their results, for comparison purposes. The GUI is also key for demonstrating the potential that optimization techniques have for planning system upgrades to defend against potential attacks. The new OD GUI is more flexible and user-friendly, and the new ODs can also help capture the details of system restoration following an attack, that is, how load shedding diminishes over time (days to weeks to months) as the system is repaired.

We have integrated the enhanced OD GUI into VEGA 1.0 to yield VEGA 2.0, which represents a substantial step towards a more comprehensive decision-support system.

I. INTRODUCTION

The United States' electrical power system is critical to the country's economy and security. The system's vulnerability to natural disasters or physical attacks has been recognized, but this vulnerability has been increasing in recent years because: (a) Infrastructure has not expanded as quickly as demand has, thereby reducing the 'cushion' available when system components fail, and (b) the probability of terrorist attacks has increased. [Salmeron et al., 2003-II]

The U.S. Department of Justice, Office of Justice Programs and Office of Domestic Preparedness is currently supporting the development of an integrated system to analyze the vulnerability of electrical power grids to terrorist attacks [Salmeron et al. 2003-I]. As part of this effort, research on optimization models is being carried out, and improved models are being integrated into a decision-support system called VEGA (Vulnerability of Electrical Power Grids Analysis). The first prototype of this system, VEGA 1.0, is described in VEGA [2003].

VEGA 1.0 comprises a graphical user interface (GUI), a supporting relational database (DB) and optimization tools. The GUI helps prepare power-network data for analysis, and then displays analytical results, by enabling easy navigation through customized tables and graphics containing problem data and results. This gives a user easy access to the mathematical analysis of a problem even if he or she is not an expert in mathematical modeling and optimization.

VEGA 1.0's core is an optimization model that assesses the maximum possible disruption a network might experience from a terrorist attack. (Eventually, VEGA will directly plan for reducing the consequences of interdiction by selecting cost-effective protective measures.) Naturally, this core can operate as independent entity, and its operation is, in fact, transparent to a user.

The GUI and DB are the keys to organizing planning data, reducing clerical error through embedded validations, completing missing details, filtering information according to users' needs, and displaying multiple scenarios with their results, for comparison purposes. The GUI is also the key for

demonstrating the potential that optimization techniques have in this important area of research, and enables decision-makers in the government and electric utilities to work with our models in a setting they can easily understand.

The thesis focuses on creating an enhanced GUI (and supporting DB, as needed) of the existing VEGA 1.0. The upgraded prototype refines and extends multiple features, becoming the VEGA 2.0 system.

The most important improvement that VEGA 2.0 provides is an advanced graphical representation of One-Line Diagrams (ODs). An OD is a graphical representation of an electric power system that displays three-phase interconnections with a single line, along with transformers, generating units, buses and other electrical devices and protective equipment [Chan 1990]. One of the first windows-based user-friendly tools that incorporates ODs is described by Overbye et al. [1995]. The OD representation of electric power grids in VEGA 2.0 is called an "OD GUI."

The OD GUI In VEGA 2.0 is more flexible than the one in VEGA 1.0 and improves the user's ability to interact with the graphical display, both in terms of data input and result output. The new ODs can also capture the details of system restoration, over time, following an attack.

VEGA 2.0 conforms to Windows GUI and DoD Human-Computer Interface standards [DOD, 2002] to a large extent, with extensive use of menus, controls, toolbars, and secondary windows, using either mouse or keyboard navigation. The enhanced OD GUI arranges screens, menus, and objects in general, to make the navigation process simpler and more logical.

The following provides an outline of the remainder of this thesis: Chapter II reviews the existing VEGA 1.0 environment, focusing on the system GUI and database; Chapter III addresses an audience with a programming background who may be interested in understanding the objects and data structures behind VEGA 2.0's OD GUI. A detailed description of the implementation includes grid component reshaping, explicit depiction of power flows, dynamic user input, etc. Objects and functions that are new to VEGA are illustrated; Chapter IV provides

a detailed, user-oriented description of the OD GUI, concentrating on the most important features of the enhanced ODs and their use in analysis. The chapter also includes certain technical details for those interested in the more specialized aspects of the implementation (which are not included in Chapter III because they require additional understanding about how the user and the OD GUI interact); Chapter V contains conclusions and recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II.   BACKGROUND ON VEGA

The purpose of this chapter is to provide the reader with an overview of the existing VEGA system, VEGA 1.0 [VEGA 2003]. We focus on the system GUI and DB, but the overview is incomplete without a brief description of the underlying interdiction model and the power-system components that are represented in that model. We begin with that description. (See Salmeron et al. [2003-II] for more detail.)

A.   INTERDICTION PROBLEM

Electric power grids are complex systems comprising thousands of electric devices such as generating units, transmission lines and distribution lines. The "backbone" of the grid is the high-voltage transmission network, typically consisting of lines rated at 69 kV or above. Since this network carries large amounts of energy over long distances in a concentrated fashion, it is an attractive target for terrorists to attack. That is, the disabling of just a few key components in this network could cut off electrical service to huge numbers of customers over a wide geographic area. (This contrasts with a low-voltage distribution network that connects customers to the transmission network, and which covers a comparatively small area; the disruption of such a network would have a relatively modest effect.)

Mathematical models in Salmeron et al. [2003-II] describe the problem of optimally disrupting an electrical transmission network by selectively interdicting (attacking) a limited number of its components. (Ultimately, models will be developed to harden a network against interdiction, but the first step in doing this is to understand optimal interdiction.) The interdictor's model seeks to use limited interdiction resources to maximize "disruption" which is the amount or cost of the energy shed. Disruption is measured by solving one or more optimal power flow problems given the network's post-interdiction configuration or configurations. (The network will typically experience a sequence of

5

configurations as interdicted components are repaired or replaced over time.) An algorithm to (approximately) solve this bi-level max-min problem is also described. The models and algorithm are implemented as a module of the VEGA system. The following briefly defines the general concepts related to electric power networks and interdiction as used in VEGA. (Some of these definitions have been obtained from the Energy Information Administration [EIA 2003] and Elec-Saver [2003].)

Transmission: The movement or transfer of electric energy over an interconnected group of lines and associated equipment between points of supply and points at which it is transformed for delivery to consumers, or is delivered to other electric systems. Transmission is considered to end when the energy is transformed for distribution to the consumer.

Transmission System (also referred to as Electric Power Grid, Electric Network and Electric System in this thesis): An interconnected group of electric transmission lines and associated equipment for moving or transferring electric energy in bulk between points of supply and points at which it is transformed for delivery over a lower-voltage distribution system to consumers, or is delivered to other electric systems.

Generating Unit (or Generator): Any combination of physically connected generator(s), reactor(s), boiler(s), combustion turbine(s), or other prime mover(s) operated together to produce electric power.

(High-voltage) Transmission Line: The high-voltage ($\geq$ 69 kV in the U.S.) conductors used to carry electrical energy from one location to another.

Bus (or Busbar): A heavy, rigid electrical conductor that makes a common connection between several electrical circuits.

Transformer: A static electrical device which, by electromagnetic induction, regenerates AC power from one circuit into another and/or changes the voltage of alternating current.

6

Load: Demand for electricity (MW) at a specific point in time.

Consumer Sector: A type of load with specific requirements (e.g., amount of power demand and cost for failing to provide it),

Substation: Facility with equipment that switches, changes, or regulates electric voltage and current.

Interdiction Resource: A numerical value associated with a mathematical expression that represents the capacity of terrorists to carry out attacks. For example if such an expression has the form: "(3 × total number of attacks to buses) + (1 × total number of attacks to lines) ≤ 5," then "5" is the interdiction resource.

Scenario: A particular value of the interdiction resource. In VEGA, we analyze the worst-possible Interdiction Plans (see below) for user-selected scenarios.

Interdiction Plan: Specific subset of the electric system equipment that might be interdicted by terrorists. Optimal or near-optimal interdiction plans are identified in VEGA for given interdiction-resource scenarios.

Power Shed: Amount of power (MW) that cannot be supplied to a load or loads (one or several customer sectors) at a specific point in time.

Energy Shed: Amount of energy (MWh) that cannot be supplied to the load (one or several customer sectors) over the course of a given time period.

Disruption: The cost of power shed in dollars/hour or the cost of energy shed in dollars, as a consequence of implementing an interdiction plan.

Period (of Restoration): Each of the stages that an electric power network undergoes following an attack, as interdicted components are repaired or replaced over time.

One-line Diagram (OD): Schematic drawing of an electrical power system that uses graphical symbols to represent electrical equipment such as buses, generators, loads, transmission lines and transformers. It may incorporate

7

numerical values for the system, such as line power flows, generating unit outputs, bus voltages, etc. We also adapt our ODs in VEGA to represent interdicted equipment.

Optimization Parameters: Input data to control the interdiction-optimization process within VEGA. These data include the maximum number of iterations for the module's algorithms and the solver engine to be used, among others.

Case: A set of network data to be analyzed, along with the results of the analysis. Each case covers a single power grid, including all physical data (i.e., network data such as line impedances, generating capacities, etc.), non-physical data (e.g., interdiction resource, optimization parameters, etc.) and results. In VEGA, we can create a new case instance, import a case from external sources, save a case, open an existing case and delete a case. All of the data and/or results are lost when the user makes modifications without "saving as" a different case first. A case can have one or several scenarios as part of the analysis, and all of these are still considered as part of the same case.

B.   VEGA 1.0

1.   Application Overview

VEGA is an integrated decision-support system comprising a GUI, a relational DB management system (RDBMS), an optimization module and an administration program that controls the aforementioned components (Figure 1). VEGA 1.0 is the first version of this system and has been built on the Microsoft (MS) Windows 2000 operating system [Microsoft 2003].
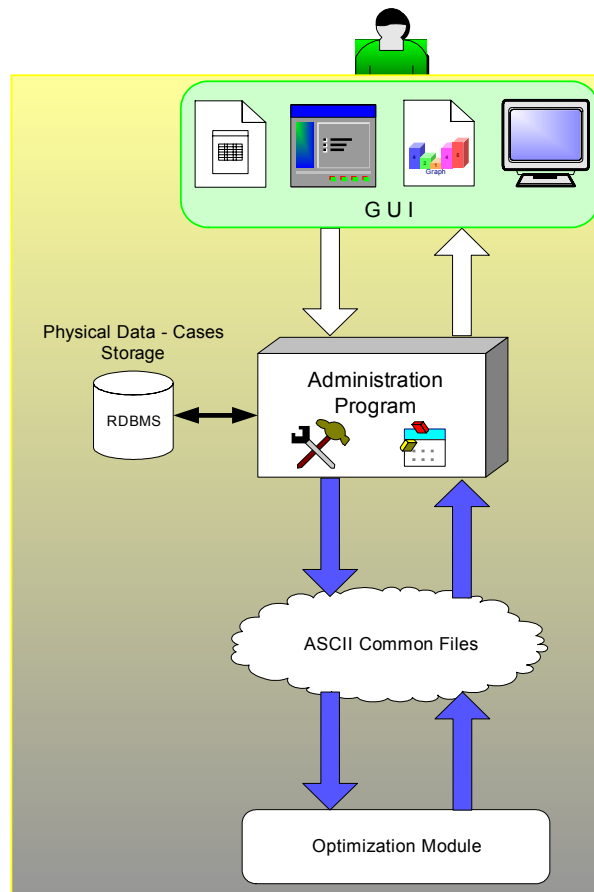
Figure 1.   Overview of the VEGA system. The system's core is an administration program responsible for the interconnections and the exchange of information between the GUI, the Relational DB Management System (RDBMS), and the optimization module. The last two modules are transparent to the user.

The administration program and the GUI are implemented in the MS Visual Basic (VB) 6.0 programming language [Microsoft 1998, 2003-I], [Balena 1999] supported by a RDBMS implemented with MS Access 2000 [Microsoft 2003-II].   The underlying optimization module is implemented using GAMS [GAMS 2003, Brooke et al. 1996]. Data transfer and synchronization of the GUI with GAMS are performed by means of plain ASCII files, because GAMS is not available as a callable or dynamic library; therefore, GAMS executes as an external program.

The front-end application responsible for the VEGA GUI uses a Windows-based methodology that facilitates for the user:

- Network data input,

- Other data input, including possible scenarios, optimization parameters, etc.,

- Analyzing results provided by the optimization model,

- Graphical display of the network, input data, and output results, and

- Administration of multiple cases with several scenarios per case.

Figure 2.   Examples of Tabular Data Representation in VEGA 1.0. The table in the foreground contains data for the electric system buses, whereas the one in the background contains data for generators.

The GUI in VEGA 1.0 uses VB tables in order to import data from the database, and edit the records associated with a problem. Figure 2 shows an example of data tables for Buses and Generators.

Upon completing all necessary data entry, the user can invoke the optimization module in order to produce optimal (or near-optimal) interdiction plans. These plans can be displayed in tabular form similar to those in Figure 2. (Of course, the ultimate value of these plans is to help the user identify critical

system components. It is these components that must be hardened, protected or backed up by spares, to reduce the potential for large disruptions.)

VEGA 1.0 can also display ODs (Figure 3) but they have limited design capabilities, they lack some basic features such as object mobility, zooming and printing, and their objects are not constructed following appropriate design standards.
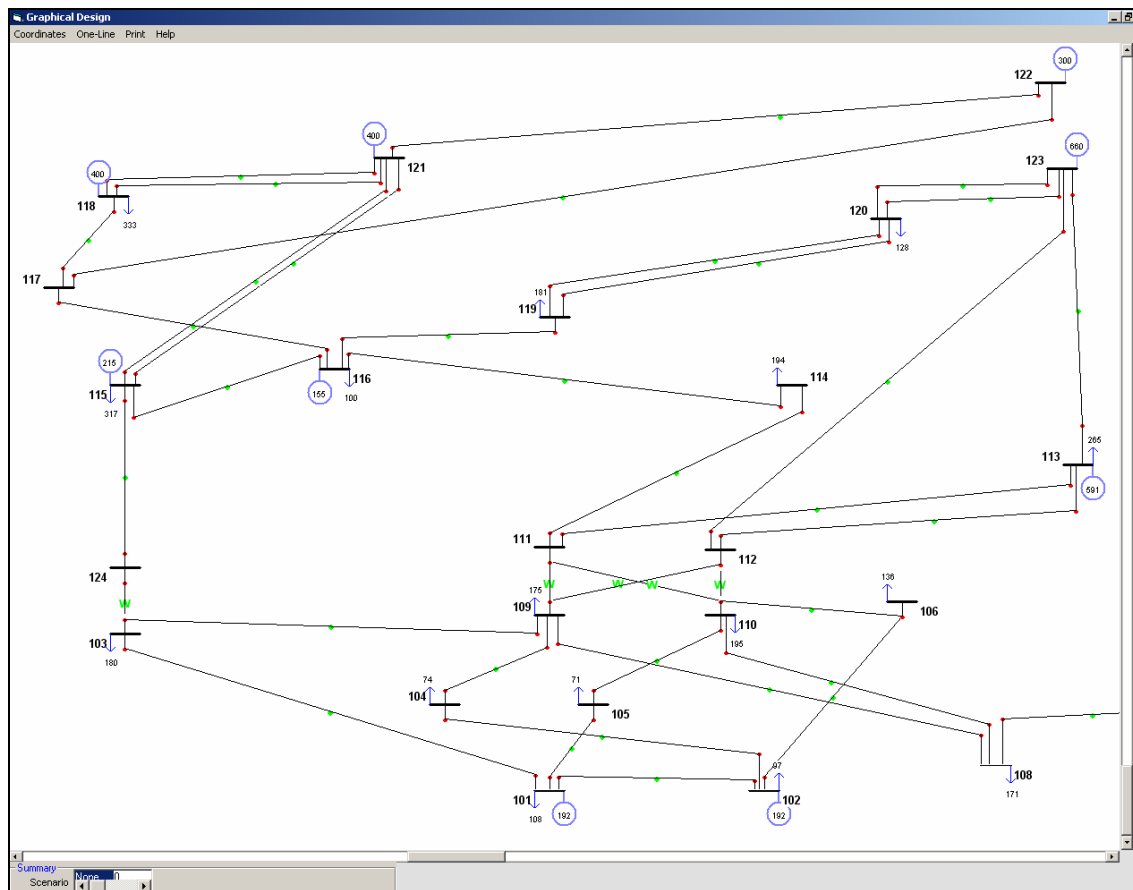
Figure 3.   Example of an OD in VEGA 1.0. Buses (heavy black lines), loads (blue arrows), generators (blue circles) and labels are designed with little mobility. There are no dialogue windows to allow user interaction and direct access to local data or results (i.e., for a specific component). Zooming is poorly designed: objects are not actually zoomed, but spaced away from (or made closer to) each other. Other features to increase usability, such as printing or exporting the OD to other applications are not implemented.

2. Database Management System

The underlying relational DB of VEGA 1.0 consists of tables and relationships. Tables contain the data for the incumbent electrical network, along with optimization results (if any) produced by the optimization module. The relational structure ensures data uniqueness and data integrity and performs part of the necessary validations. The DB allows scaling the application to fit problems of different dimension.

The tables that hold the initial information of a case and parameters for the optimization model are:

Table Name	Description
Bus	Physical data for each bus and graphical data for the representation of the bus in the OD
Bus_Sector	Data for each customer sector at every bus
Generator	Physical data for each generating unit
Line	Physical data for each line and each transformer and graphical data for the representation of the line in the OD
Sector	Physical data for each customer sector
Substation	Physical data for each substation
Opt_Parameters	Optimization parameters
Plan	General data about the incumbent plan or case

Table 1.　VEGA Data Tables.

Using the data in Table 1, VEGA's optimization model (Salmeron et al. [2003-II]) computes an "optimal interdiction plan," i.e., an interdiction plan that maximizes disruption, for every specified interdiction-resource scenario. . All these plans are stored and can be displayed for comparison purposes.

The effects of an interdiction plan change as the grid's interdicted components are repaired over time. This time can be broken down into Periods during which the level of disruption is may be assumed to be constant. The tables used in the data model to store the results for every Scenario and for each time Period:

- Scenario
- Scenario_Period
- Scenario_Sector_Period
- Scenario_Substation
- Scenario_Substation_Period
- Scenario_Bus
- Scenario_Bus_Period
- Scenario_Bus_Sector_Period
- Scenario_Generator
- Scenario_Generator_Period
- Scenario_Line
- Scenario_Line_Period

Each of these tables contains results at the specified level of detail. For example, Scenario_Bus_Period contains results for every bus during each possible period of restoration for each scenario. Typical results at this level for the given scenario are whether the Bus is "in service" or not (during the period), as well as the power generated and/or consumed at the bus.

VEGA uses the data tables and result tables to interact with the user. VEGA 2.0, described in the following chapters, uses the same tables as VEGA 1.0.

Figure 4 depicts the overall relational structure of the database.
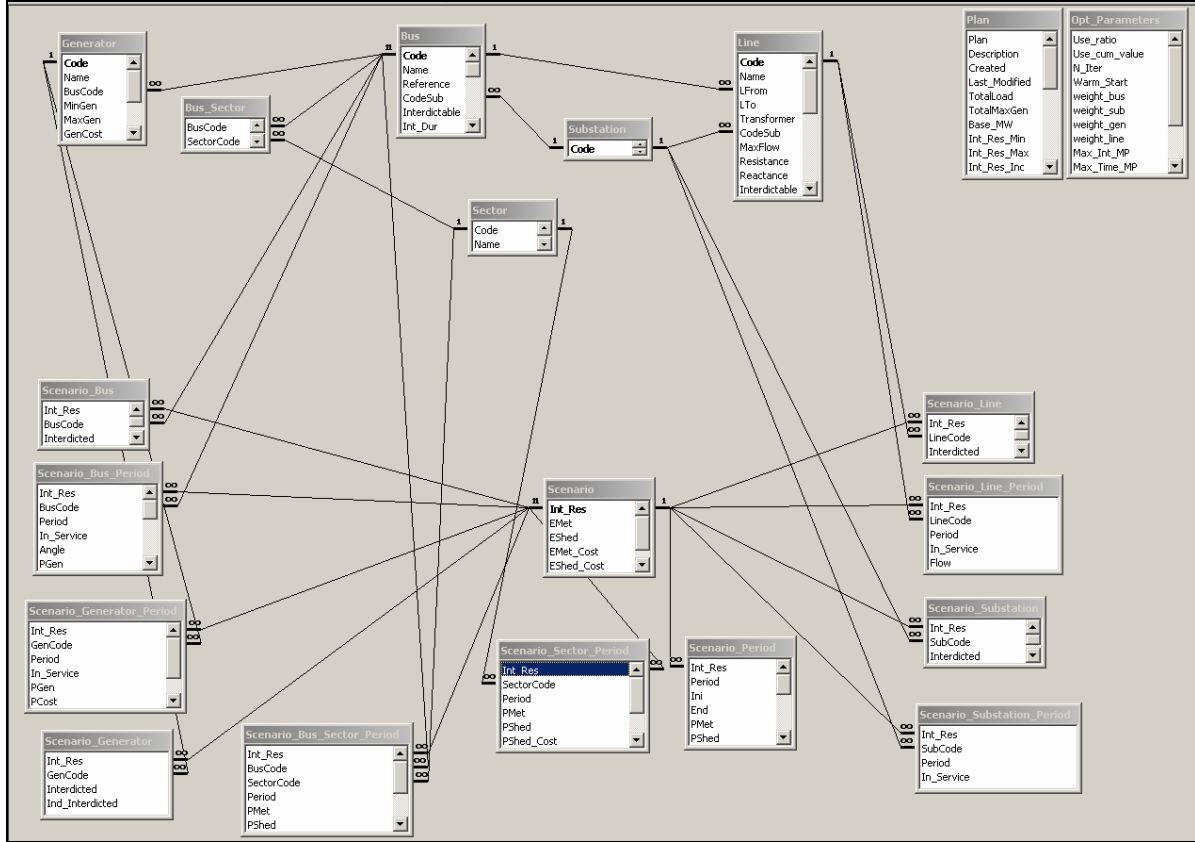
Figure 4.   VEGA RDBMS Table Structure and Relationships.

III. ONE-LINE DIAGRAMS FOR VEGA 2.0

This chapter addresses an audience with a programming background who may be interested in understanding the underlying objects and data structures behind VEGA 2.0's ODs. A detailed description of the implementation includes grid component reshaping, explicit depiction of power flows, and dynamic user input, among others.

A. INTRODUCTION

It is important for the reader to realize the variable and typically large amount of data that VEGA must handle. Every case under analysis has a different number of buses and lines (thousands of each in real cases), and transformers (typically hundreds) [e.g., PowerWorld 2003, University of Washington 2003]. Each bus also has a different number of generators and customer-sector loads for each case under analysis. All these network components have different behavior by scenario and period within each scenario. This complex structure is determined "on the fly," so that a user can create and/or modify a network's description at any time.

The requirement for data scalability, alluded to in the previous paragraph, has been satisfied in VEGA 2.0 by designing and coding a runtime object-oriented graphical data model following the DB model that was briefly introduced in Chapter II, Section B.2.

It is worth noting that, although VB is an excellent language for creating comprehensive GUIs, it lacks proper tools for creating the complex graphical structures that VEGA requires. In order to overcome the graphical shortcomings we have incorporated a third-party Microsoft ActiveX control [Microsoft 2003-II]. The data model for the OD and the new ActiveX control are described in the following sections.

B.   DATA MODEL ARCHITECTURE FOR THE OD

The data model for the advanced graphical representation of an OD consists of native VB data types and User-Defined Types (UDTs). A UDT is a compound data structure that holds several variables of simpler data types. For that reason, UDTs are often used to represent database records, which also consist of a number of related components of different data types.

For purposes of efficient execution, all the UDTs are created and stored in a random-access memory data structure, which is an array of UDTs.   To minimize memory consumption, array sizes change dynamically during runtime to accommodate all (but not more than) the necessary case data.

As a result, the OD data model consists of data objects that encapsulate all the necessary information provided by the database, which may include results if the optimization process has executed successfully. This allows each data object to be the sole owner of its data. Every data object is created and stored in a dynamically sized array at the beginning of the OD GUI execution. One of the advantages of this data architecture is that data are stored in a memory area that can be accessed directly through VB code by just supplying appropriate array names and indices (see below).

The main UDTs, BusData, SectorData, GenData, and LineData/TransData/TransLineData) are defined as *Public* in separate VB modules, called Buses, Sectors, Generators and Lines, respectively.   These UDTs can be used as arguments for Private and Friend procedures defined in any type of module in the VB project for the VEGA GUI. They can also be used as arguments in Public procedures defined in any VB modules of the application, although not in other types of modules such as forms, MDI forms or user controls.

There is one module for each main UDT. The secondary UDTs (Scenario, Period, Point) are defined as *Private* inside each main UDT module. With this design, we can use the same name for secondary UDTs containing different elements, but using the same reference style for all the main UDTs.   For

example, consider how this architecture is used to retrieve information from (or to pass data to) the OD data model:
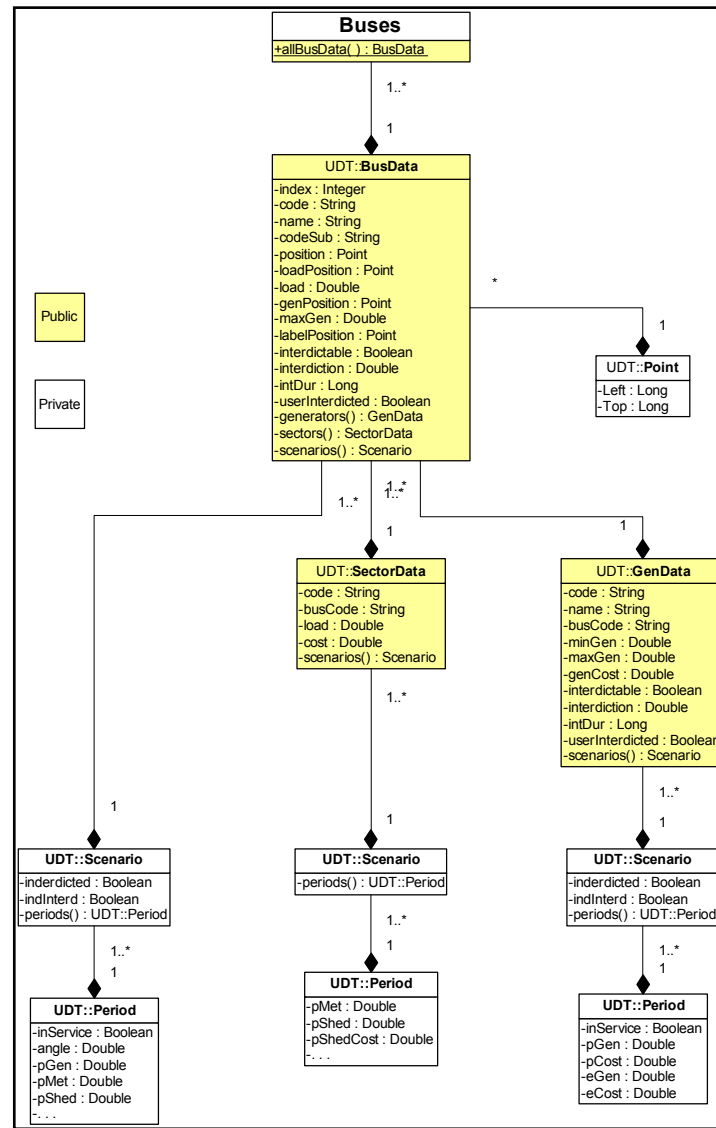
Figure 5.   BusData, SectorData, GenData UDTs. Hierarchy is represented with a Unified Modeling Language diagram. Each subordinate UDT may exist one or more times (1..*) as a part of a hierarchy-higher UDT. For example a Period UDT may exist one or more times as a part of a Scenario UDT.

Assume that we would like to refer to a given data attribute called "angle" associated with period 1 in scenario 5 for generator 1 of bus 7. We would use the following code for that purpose:

```
allBusData(7).generators(1).scenarios(5).periods(1).angle
```

17

Likewise, the reference for attribute "pMet" associated with period 1 in scenario 5 for sector 1 of bus 7 would be:

```
allBusData(7).sectors(1).scenarios(5).periods(1).pMet
```

Notice that we are using the same data structure even if generators and sectors have totally different scenario and period UDTs.

1. Buses Data Structure

Figure 5 shows the data structure for a system bus in the OD GUI. We next describe the role played by each UDT within the data structure.

- Point UDT

The Point defines the Left and Top position of a bus and its components (e.g., customer sectors and generating units) within the OD graphical representation on the screen. It consists of two Long VB data types where the positions are stored in twips. (A *twip* is 1/20 of a printer's point; 1,440 twips equal one inch). By default, all VB movement, sizing, and graphical-drawing use a unit of one twip. These measurements designate the size an object will be when printed. Actual physical distances on the screen vary according to the monitor size.

- BusData UDT

The BusData UDT consists of a number of VB data types, four Point UDTs for positioning bus components on the user's screen, and three dynamically created arrays of Scenario, SectorData and GenData UDTs. Each BusData UDT contains all the information for a specific Bus in the electrical network, such as identification characteristics, location, power flows and interdiction behavior.

- Period UDT

A Period UDT consists of a number of simple VB data types, which depend on the associated electrical network component. For example as we can see in Figure 5, the Period UDT associated with BusData UDT has different

elements than the Period UDT associated with GenData UDT, because they play a different role in an electrical network.

- Scenario UDT

A Scenario UDT consists of two simple VB data types and one dynamically created array of Period UDTs. With this structure, we ensure each scenario has the exact number of periods required to accommodate the results that depend on the specific time-frame of the scenario.

- SectorData and GenData UDTs

Each of these UDTs consists of a number of simple VB data types and one dynamically created array of the Scenario UDT. The associated sector and generator have their own user-defined internal data along with the solution-result data once the model is successfully optimized.

2. Lines/Transformers Data Structure

The Lines/Transformers data architecture (see Figure 6) is similar to the Bus data architecture. The only difference worth noting is the fact that the Lines/Transformers data structure contains three separate and dynamically created arrays: One for use by the line UDTs (allLineData), another for the transformers UDTs (allTransData) and one special UDT for lines that connect buses to transformers (allTransLineData). The reason for using three UDTs is the different behavior of the objects they represent. (This will be clarified when the Transformer object and the TransLine object are described in Section D).
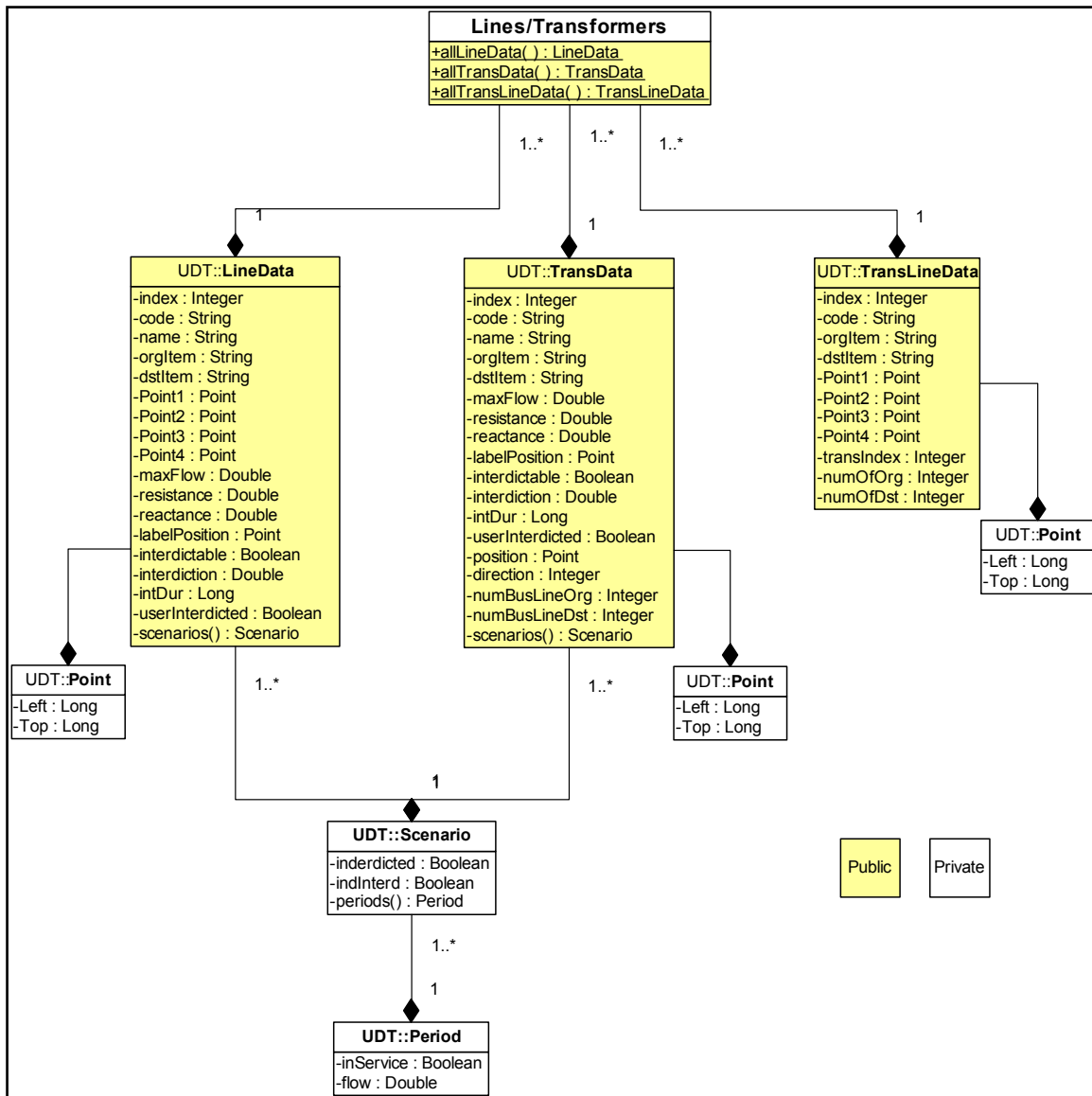
Figure 6.   LineData, TransData and TransLineData UDTs. Hierarchy is represented with a Unified Modeling Language diagram. Each subordinate UDT may exist one or more times (1..*) as a part of a hierarchy-higher UDT. For example a Point UDT may exist one or more times as a part of a LineData UDT.

C.   ADDFLOW ACTIVEX CONTROL

1.   General Description

In order to overcome the limitations of the VB language for graphical design, we use a third-party Microsoft ActiveX control called AddFlow (version 4.2) [Lassalle 2003] instead of the VB native graphical tools. This control has

been implemented by Lassalle Technologies using Visual C++ 6.0 [Microsoft 2003-II] and is based on the Microsoft Foundation Class.

A Microsoft ActiveX control [Microsoft 2003-II] is a set of technologies that enable software components to interact with one another in a networked environment, regardless of the language in which the components were created. Currently, ActiveX is used primarily to develop interactive content for the World Wide Web, although it can be used in desktop applications and other applications. ActiveX controls can be embedded to produce animation and other multimedia effects, interactive objects, and sophisticated applications.

The AddFlow ActiveX control allows us to create instances of so-called AddFlow objects, which are actually complete diagrams that can be added to any VB form.

2.   AddFlow Object

The AddFlow object is the container of one or more Node objects that can be coupled to each other by Link objects. Each time a Node is created, it is stored inside an object collection called Nodes where Line objects are stored inside a collection called Lines. The visible part of the Addflow object is the AddFlow diagram. Figure 7 shows a typical three-node three-link example of such a diagram.
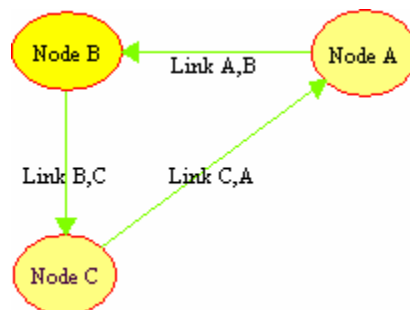
Figure 7.   Addflow Object Example. Three Node objects ("Node A," "Node B" and "Node C") are connected to each other by three Link objects ("Link A,B", "Link B,C" and "Link C,A").

21

Nodes are the basic objects of an Addflow diagram, and Links are meant to connect Node objects. Both Node objects and Link objects have many attributes that can be changed at design- and/or run-time, such as color, text, drawing style and font.

Nodes may be moved or resized whereas Links may be stretched and divided into several segments in order to follow the route the user decides. For consistency, a Link cannot exist without its origin and destination nodes. If either of these two Node objects is removed, any associated Links are also removed. For instance, if we remove Node C from the example of Figure 7, the Addflow object automatically deletes the two links associated with Node C, namely links C,A and B,C; see Figure 8.
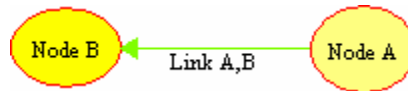
Figure 8.   Deleting AddFlow Objects. The two-node, one-link diagram is the result of deleting Node C from the diagram in Figure 7. Node C's deletion also causes the deletion of Link B,C and Link C,A.

3.   Node and Link Objects

A Node object is created by using the Add method of the Nodes collection, which stores the list of all Nodes in a given AddFlow object. The Add method allows the programmer to specify both the position and the size of the Node that is being created. The Node object type name is *afNode*.

For example, the following code creates a Node object by first instantiating it and then adding it to the Nodes collection:

```
Dim nBus As afNode
Set nBus = addFlow.Nodes.Add(position.Left, position.Top, iHeight, iWidth)
```

When it is created, a Node receives default values for most of its attributes. Some of the Node attributes are:

- Size: Height, Width.

- Colors: Drawing color; Text color; Fill color.

- Drawing pen: Style; Width.

- Shape: Allowed shapes are rectangle, ellipse, etc.

- Text: Alignment; Autosize mode; Transparency; Font.

For example, if we want to create a black circular Node, we can do that by setting the Shape property to "afEllipse," and the DrawColor property to "vbBlack." The circle shape is produced by setting Node.Height equal to Node.Width. Additionally we can associate text, a picture or user data to be displayed with the Node.

Nodes are collected into the Node collection. Each Node has two collections of Link objects. One for the outgoing Links (OutLinks) and another for the incoming Links (InLinks). Which Link objects are considered outgoing or incoming for a Node is decided at the time the Link is created.

A Link object allows linking two Nodes. It is created programmatically by using the Add method of the OutLinks or InLinks collections in order to associate the Link with the Nodes. By construction of the Link object, we establish which Node is deemed the origin (or destination) of the Link. The other Node object in the declaration of the Link automatically becomes the destination (or origin) Node of the Link. To accomplish this, we directly create the Link as part of the OutLinks collection of the origin Node, which automatically adds the Link to the InLinks collection of the destination Node. Alternatively, we can create the Link as part of the InLinks collection of the destination Node, which automatically adds the Link to the OutLinks collection of the origin Node (see code below). OutLinks and Inlinks collections serve the equivalent purpose for Link objects as the Node collection for Node objects. The Link object type's name is *afLink*.

The following code creates a Link object (identified as "line") by first instantiating and then adding it to the OutLinks collection of the origin Node

(identified as "node1"). This "line" is also automatically added to the Inlinks collection of the destination Node (identified as "node2"):

```
Dim node1 As afNode
Dim node2 As afNode
Dim line As afLink
Set line = node1.OutLinks.Add(node2)
```

Graphically, this code draws a line from the so-called "origin" Node object to the "destination" Node object. In the previous example, "node1" is the origin and "node2" is the destination Node. Alternatively, the following code produces the same result as above, because we add the Link object "line" to the InLinks collection of "node2":

```
Dim node1 As afNode
Dim node2 As afNode
Dim line As afLink
Set line = node2.InLinks.Add(node1)
```

When it is created, the Link also receives default values for most of its attributes. Some of the Link attributes are:

- Colors: Drawing; Text

- Drawing pen: Style; Width

- Arrow shape: ArrowOrg; ArrowDst; ArrowMid

- Text: Font

A Link may have several segments in order to represent graphically the route the user wants the link to follow. Graphically we can identify segments by locating "elbow points" on a Link object. For example in Figure 9, the Link object between the two Node objects has two segments. The default behavior is that

the first segment is directed toward the center of the origin Node object and the last segment is directed towards the center of the destination Node object.

Every Link object has a "LinkPoints collection." A LinkPoints collection has at least two elements (called "linkpoints") which graphically represent the ending points of each segment in the Link. For instance, if a Link has three concatenated segments, its LinkPoints collection contains four points. If we delete the LinkPoints collection by using the *clear* method, then the number of points is reset to two, and the Link becomes a straight, one-segment Link.

The ability to add or remove segments from a Link object is enabled only if the LinkStyle property of the Link object has set to afPolyline. Otherwise, the Link object has a constant number of segments depending on the LinkStyle property. For convenience, all the Links in the OD GUI have the LinkStyle property set to afPolyline (i.e., we may add or remove points to that collection at run-time), although, as we shall see later in this chapter, the user will not be allowed to increase the number of Link segments to more than three.
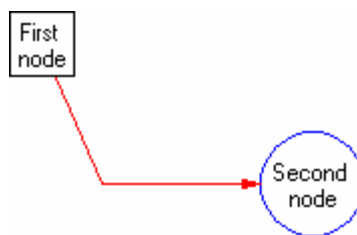

Figure 9.   AddFlow Diagram Example.   We connect a black rectangular node named "First node" to a blue circular node named "Second node" through a red elbowed link consisting of two segments and three linkpoints (the elbow-point and the two segment-ending points).

The following self-commented code is an excerpt from a VB program which, at some point, creates the diagram in Figure 9:

```
' Create two nodes and link them.

Dim node1 As afNode, node2 As afNode
Dim link As afLink

' Define a default fill color for next created nodes and links
AddFlow1.DrawColor = RGB(0, 0, 0)  ' Default Drawing color = Black
' Define a default shape for next created nodes
AddFlow1.Shape = afRectangle  ' Default shape = rectangle

' Create a blue rectangular node and associate a text to this node
'   .Add(left, top, width, height)
Set node1 = AddFlow1.Nodes.Add(100, 100, 500, 500)
node1.Text = "First node"

' Create a blue elliptic node and associate text to this node
Set node2 = AddFlow1.Nodes.Add(2000, 1000, 800, 800)
node2.DrawColor = RGB(0, 0, 255)   ' Blue color node
node2.Shape = afEllipse
node2.Text = "Second node"

' Create a red link between node1 and node2
Set link = node1.OutLinks.Add(node2)
link.DrawColor = RGB(255, 0, 0)    ' Red color link
' Add an elbow point with specific coordinates
link.ExtraPoints.Add 800, 1400
```

4.   Useful Features

The Appendix describes the features of Addflow objects used most frequently during the implementation of the enhanced OD GUI.

D.   VISUAL REPRESENTATIONS OF OBJECTS IN VEGA 2.0

Every OD in the new OD GUI implemented in VEGA 2.0 comprises a set of objects of type Bus, Line, Transformer and TransLine. We next describe each of these object types in detail.

1.   Bus Object

The most important and complex object of the OD is the Bus object (Figure 10). It is created during runtime by using five AddFlow Node objects and six Link objects, each one with its own independent properties and methods. The Bus object is the container Node of all these objects.

26
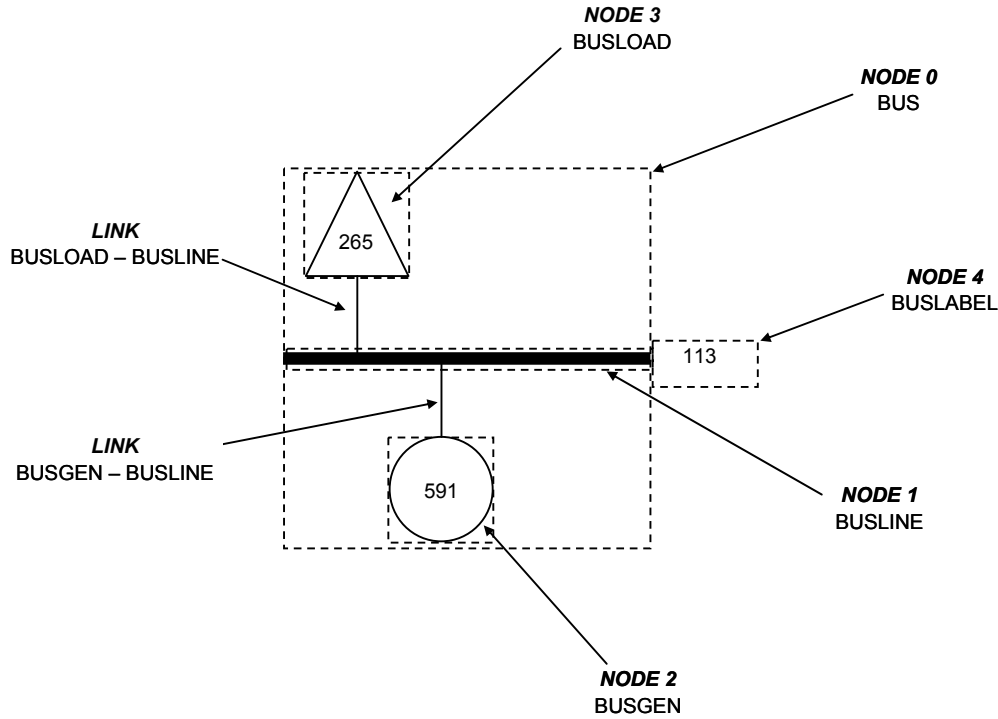
Figure 10.   The Bus Object.   Every dashed box represents a Node object.   The numbering for nodes (0 to 4) indicates the order in which Nodes are created. The two Link objects are used to convey the visual effect that the Load and the Generator objects are anchored to the BusLine.

It is important to note that, for purposes of graphical representation, multiple loads and generators connected to the same Bus are aggregated into a single load and generator, respectively. If no load is connected to the bus, none is displayed (the corresponding object still exists but becomes invisible), and likewise for generators.

The identification of Nodes in a Bus object has been controlled by means of a numerical index that characterizes the role the Node object plays within the Bus Object. These Nodes (sometimes referred to as "subnodes" given their dependence on the Bus Object) are explained in detail in the following paragraphs.

27
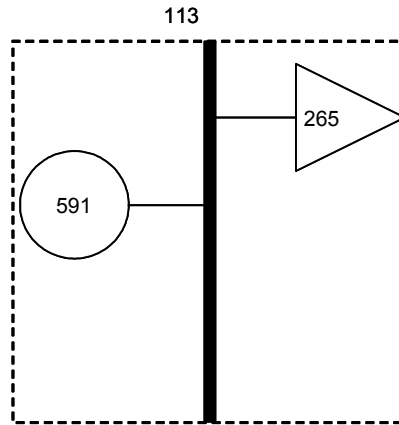
Figure 11.   Vertical Bus Representation. The Bus in Figure 10 has been rotated 90 degrees clockwise to be positioned vertically.

a.   *Bus Node (Bus)*

The Bus Node is an invisible rectangular Node that can be considered the core of the Bus object. It is used as a container for the construction of the other Node and Links of the Bus object. This is achieved by linking the center of the Bus node with all the other Nodes with rigid, non-selectable and hidden Links. Every movement of a Bus Node "pulls" its subnodes along with it, which creates the desired visual effect for label, the load and the generating units to be anchored to the Bus.

Another useful result of this design (for programming purposes) is the special connectivity rendered by Links between the Bus Node and the other subnodes of the Bus object. The Bus Node is always the origin (.org) of the OutLinks, and the other subnodes are the destination Nodes (.dst) (Figure 12).

For example, assuming we know the Bus object ("nBus") to which a BusLabel object ("nBusLabel") belongs, we will use the following VB expression to refer to the BusLabel,

```
Set nBusLabel = nBus.OutLinks.Item(4).dst
```

where the index "4" refers to the fourth item in the OutLinks collection of Node "nBus." That item is precisely a Link object (Bus – BusLabel, Figure 12), whose destination Node is a BusLabel object identified as "nBusLabel."

28
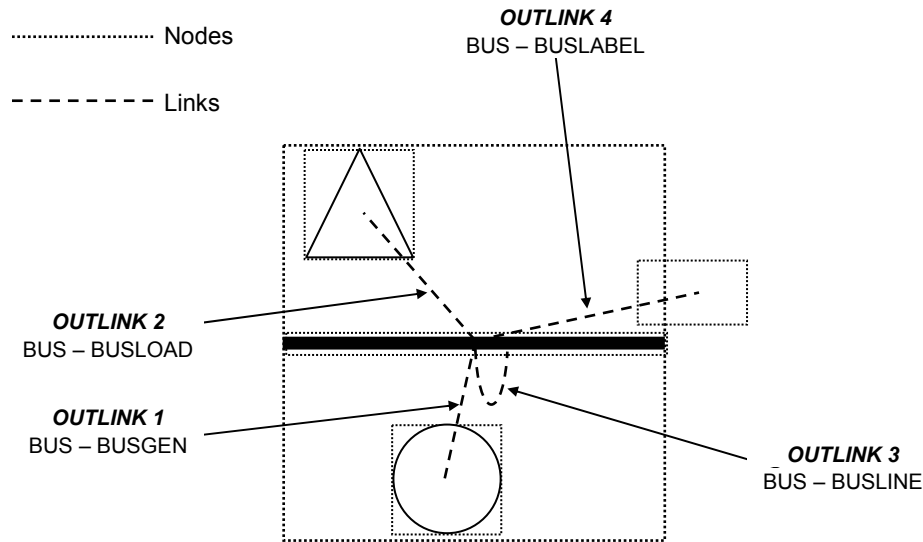
Figure 12.   Bus Object Connectivity.   Invisible Links (dashed lines) connect the center of the Bus Node (large dotted box) to all the other sub-Nodes (small dotted boxes) in the Bus object. The Links are added to the OutLink collection of Bus Node.

This design lends itself to using properties and applying methods to multiple Nodes and Links in a Bus object in a flexible fashion.   As an example, the following excerpt from VB code shows how to paint, using a specific color, all bus components within a subset of selected Buses in an AddFlow diagram:

```
For Each nBus in AddFlow.Nodes
If nBus.Selected Then
   For Each line In nBus.OutLinks
      If line.Rigid Then              'Pick only the rigid lines
         line.dst.DrawColor = color   'Paint the destination of line
      End If
   Next
Endif
Next
```

Our OD design has been setup with Bus Nodes covering a fixed area of 1,800×1,800 twips on the initial screen.   This is a user-modifiable property.   Any *click* inside this area is an event controlled by the Bus Node object.

29

An important property of the Bus Node object is the Bus.UserData because it links every Bus object to the Bus UDT. The following example shows how this is accomplished by transferring data from the Bus UDT to VB form controls for a selected Bus, which will ultimately make use of these data:

```
busIndex = busSelected.UserData          'index number of Bus object
lblBusCode = allBusData(busIndex).code   'fill a VB Label control
txtName = allBusData(busIndex).name      'fill a VB Text Box control
```

b.   Busline Node (BusLine)

A BusLine Node is an object that represents an electric busbar. It is also the Node where Link objects representing electric transmission lines originate or terminate.

A BusLine is created as a long narrow rectangular Node, displayed horizontally in the middle of the Bus Node object (Figure 13). Graphically, it represents the actual bus symbol used in and OD. The BusLine Node has the same width as the Bus object and a height of 40 twips. It is not user-selectable; therefore any *click* on BusLine area is still an event controlled by the Bus Node object. Setting the property UserData of the Node object to "1" uniquely identifies a Node as a BusLine Node.
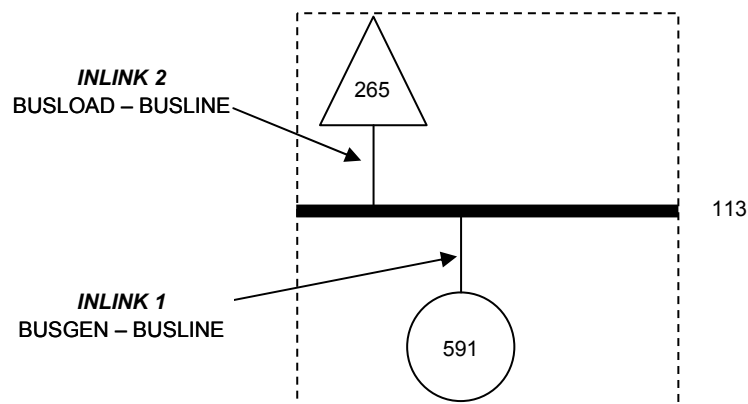

Figure 13.   BusLine Object. The generator Node (BusGen) and the load Node (BusLoad) are connected to the bus-line Node (BusLine) through two visible Links that are members of the InLink collection of BusLine Node.

30

BusLine Nodes are linked to Generator and Load Nodes with two visible, non-selectable and non-rigid Link objects. These two links are created during Generator and Load Node creation.

The following VB code shows how to refer to the BusLoad or the BusGen associated with a BusLine:

```
Set nBusGen = nBusLine.InLinks.Item(1).org
Set nBusLoad = nBusLine.InLinks.Item(2).org
```

c.   Generator Node (BusGen)

A Generator or "BusGen" Node is an object that represents one or more generating units connected to a bus. It is displayed as a circle with free movement anywhere in the Bus Node area. The movement is controlled programmatically in order to keep the BusGen within the Bus Node. Initially, a Generator Node covers a selectable area of 450×450 twips. Any mouse event triggered inside this area is an event controlled by the Generator Node. This Node type is uniquely identified through *UserData* for this subnode being set to "2."

The value of the BusGen *Text* property is displayed inside the BusGen area (e.g., "591" in Figure 13). We use this property to show the value of the total generating capacity at the bus (sum of maximum output for all generators connected to the bus) or the actual output (total power injection at the bus), depending on the display mode. If the total generating capacity at the bus is zero, the BusGen Node is made "invisible."

The BusGen Node is linked to the BusLine object with a visible, non-selectable and non-rigid OutLink.

d.   Load Node (BusLoad)

A Load or "BusLoad" Node is an object that represents one or more loads connected to a bus. It is displayed as a triangle with free movement

31

anywhere in the Bus Node area. The movement is controlled programmatically in order to keep the BusLoad within the Bus Node.

Initially, it covers a selectable area of 450×450 twips. Any mouse event triggered inside this area is an event controlled by the BusLoad Node. *UserData* is set to "3" to identify this subnode as a BusLoad Node.

As in the case of the BusGen Node, the number shown inside the BusLoad area (e.g., "265" in Figure 13) is the value of the *Text* property of the BusLoad Node. It is used to display either the total load at the bus (sum of loads from all customer sectors at the bus) or the total load that has been met at the bus, depending on the display mode. If the total load at the bus is zero, the BusLoad Node is made "invisible."

The BusLoad Node is linked to the BusLine of the Bus object with a visible, non-selectable and non-rigid OutLink.

e.   Label Node (BusLabel)

The BusLabel Label Node is displayed as a "transparent" Node with free movement not only inside the Bus Node area but also outside that area for a limited distance (this improves visualization, especially if its text is long and/or other Line objects overlap with the BusLabel).

Like the other Bus subnodes, its movement is controlled programmatically to keep the BusLabel in the desired area. The BusLabel Node covers a selectable area of 600×450 twips. Any mouse event inside this area is an event controlled by the BusLabel Node. *UserData* is set to "4" to identify this subnode as a BusLabel Node.

The text inside the BusLabel is the value of *Text* property of the BusLabel node, and it is used to display the bus code and the substation code that the bus belongs to (if any).

f. A Special Case of Label Node (BusLabel): Substations

Buses associated with a particular substation will display the substation code as part of the BusLabel code for the Bus.

Substations are not represented graphically as independent objects. The main reason for not doing so is that a substation may consist of several buses and transformers, whose relative location in the OD cannot be anticipated. This makes complicated finding a proper "Substation" object that can accommodate all possible situations. In addition, substations do not represent a major source of data and/or results:

- Each substation consists of a number of transformers and buses (which are already represented individually in the OD).

- Substation data is limited to interdiction parameters: Whether the substation is interdictable or not, required interdiction resource to attack a substation, and time to repair (if interdicted). This information is available from the GUI Data Menu.

- Substation results are limited to whether or not the substation has been interdicted, and whether it is "in service" or "out of service." This information will be represented in the OD as information associated with the Buses, as will be described in Chapter IV.

2. Transformer Object

Like the Bus object, the Transformer object is created during runtime. A Transformer object requires two Node objects and four Link objects, each with its own properties and methods (see Figure 14).

The Trans Node is an invisible rectangular Node that can be considered the core of the Transformer object. It is used as a container for the construction of the other Nodes and Links of the Transformer object, holding them together.
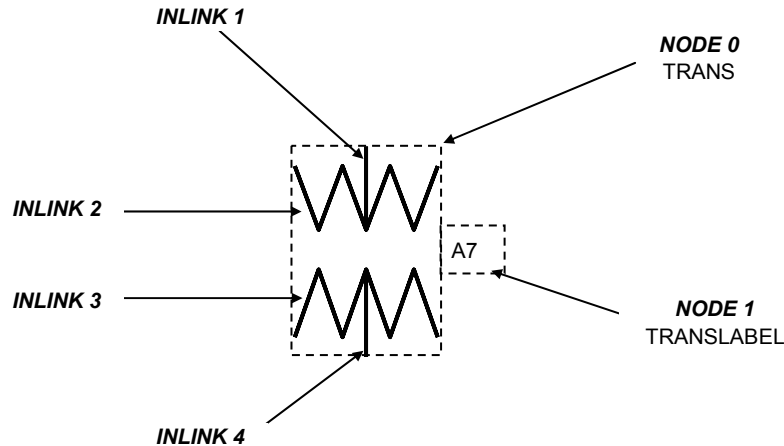
Figure 14. Transformer Object. Consists of four Link objects (used to create the familiar electrical-engineering symbol for a transformer) and two Node objects ("Trans," which contains the four Link objects, and "Translabel," which contains a label). The Link objects are members of the InLink collection of the Trans Node.

The Transformer comprises four AddFlow Links to represent the standard symbol for an electrical transformer used in ODs. These Links are rigid, visible, non-selectable, and they have specific coordinates for each LinkPoint. The TransLabel Node is linked with a rigid, non-selectable and hidden Link to the Trans Node. Every movement of the Trans Node forces all the connected objects to follow it.

The TransLabel Node is displayed as a transparent Node. It has free movement anywhere within the Trans Node area, and also outside that area for a limited distance. Similar to the BusLabel object, this movement is controlled programmatically.

The TransLabel Node covers a selectable area of 800×200 twips. Any *click* inside this area is an event controlled by the own TransLabel Node. Its *Text* property is used during runtime to display either the transformer identification code or the power flow through the transformer, depending on the display mode.

3.   Line Object

The Line object (Figure 15) is created as a Link object provided by AddFlow. Accordingly, all the AddFlow Link properties and methods are also

available for the Line object. In addition, we have implemented certain restrictions in order to customize it for its use in the OD GUI.
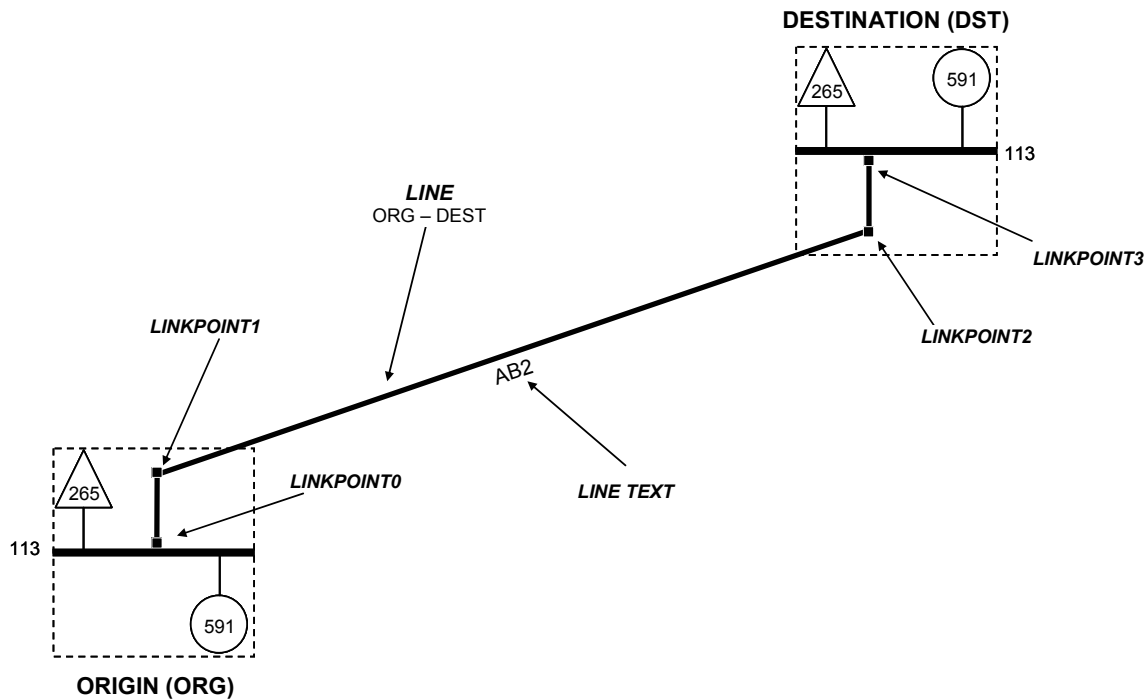
Figure 15.   Line Object. This is an AddFlow Link object with special characteristics: It comprises three segments and four extreme points called LinkPoints. The first and the third (outer) segments are connected to the second (inner) segment by two elbow-points, which are two of the LinkPoints. The two other segment-ending points are one in the first segment and another in the third segment. They connect these segments perpendicularly to the origin and destination Bus Nodes of the Link.

A Line object always connects two Bus objects. Specifically, it connects a Bus object (playing the role of origin Bus) by using a Link from any point on its BusLine Node to another Bus object (playing the role of destination Bus) at any point on its BusLine Node. (The differentiation between origin and destination Bus objects is for programming purposes only, although it follows the values entered by the user as "from" and "to" buses. This is simply a customary way to let the user decide which direction of power flow is considered positive or negative, which is important for mathematical computations.)

By default, AddFlow Link objects have multiple segments. For the purpose of representing ODs, it suffices, and it is actually more practical, for a Line to have no more than three segments. The first and the last segments are

35

always directed perpendicularly to either BusLine Node, and the middle segment connects the other two segments in whatever orientation they dictate.

A Line object has four LinkPoint objects where each LinkPoint is located at either end of any of the three segments. The LinkPoints movement is controlled programmatically so that the first and the last LinkPoints can move along the BusLine only, whereas the middle ones can move towards or opposite to the BusLine object.

4. TransLine Object

A TransLine object (Figure 16) always links one side of a Transformer object to a Bus object. It is a Link from the origin BusLine (in the origin Bus) to the destination Transformer at a certain connection point given by the coordinates of the midpoint between the edges of the Trans Node.
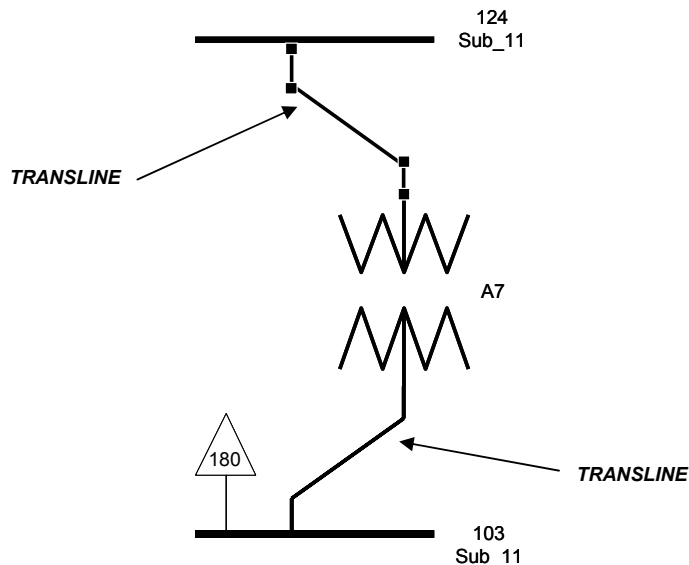
Figure 16.   TransLine Object. This is an AddFlow Link object, almost identical to the Line object.   TransLine objects connect a Bus object and a Transformer object. LinkPoints for the upper Link are shown.   The LinkPoint attached to the Transformer object cannot be moved directly, but follows the transformer (Trans object) movement.

A TransLine object has no more than three segments and is identical to a Line object except that the LinkPoint that is connected to the Transformer cannot

36

be moved. The TransLine object design is more convenient and flexible than using a straight one-segment line to connect a BusLine and a Transformer, because it allows movement along the BusLine,and also allows stretching the two ending segments perpendicularly.

E.   CONNECTIVITY WITH THE VEGA DATABASE

1.   Initial Representation

Figure 17 summarizes the main processes involved in the creation of an OD. Before an OD can be created, the user must provide X-Y-coordinates (in any reference system) for each bus in the electrical network. That is, in fact, the only input VEGA 2.0 needs in order to create an OD.

The function *NewCoordinates()* is an automated procedure (to be described in Section F) that creates screen coordinates (twips) for each network component in an OD being developed. Screen coordinates are stored in the database to allow the user to retrieve previously generated ODs and their modifications. In case that OD data already exist at the time of creating a new OD, a warning message prompts the user to confirm the new OD creation action, because that will permanently delete the previous OD and its data.
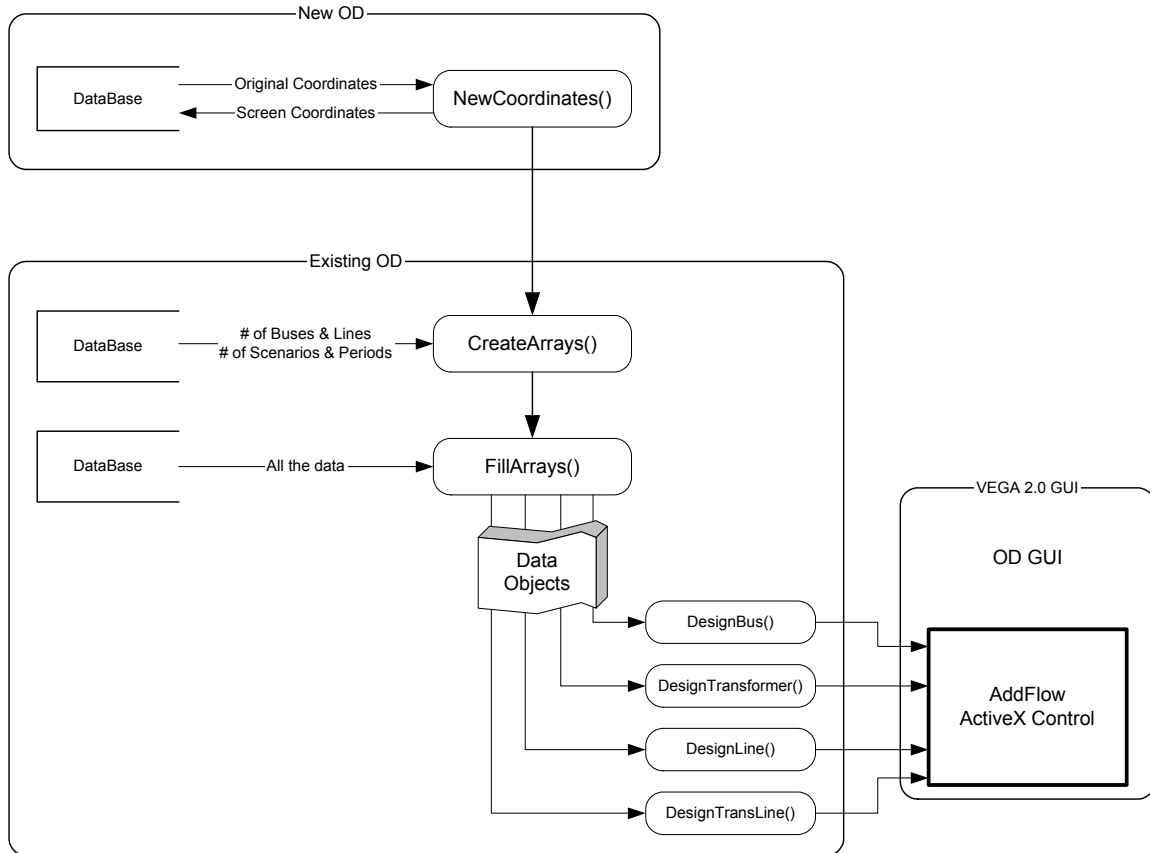
Figure 17.   Creation of ODs.   There are two ways to create an OD, from scratch using the "New OD" procedure, or by loading an existing OD saved in the DB using the "Existing OD" procedure. The "New OD" procedure creates screen coordinates for all the OD objects. Then, it invokes the "Existing OD" procedure to bring the OD up on the screen. This procedure dynamically resizes memory arrays according to the number of objects to be displayed. The procedure then fills the arrays with existing data, including object screen coordinates. OD objects are physically instantiated as AddFlow objects, and they are appropriately customized before being displayed.

If the user wants to build an OD from scratch, the OD GUI employs the "New OD" procedure, which creates the aforementioned screen coordinates and saves them into the DB. Then, "New OD" invokes the "Existing OD" procedure, which not only retrieves these coordinates from the DB, but also transfers all the existing information on the incumbent case (data and results, if any) into system memory.   Both coordinates and data are used to position electrical network objects and their label values on the screen.

The *CreateArrays()* function checks the existing number of buses, lines, scenarios and periods in the database, and dynamically creates the data

structure described in Section III.B.   The *FillArrays()* function transfers the information from the database to the previously created memory data structure.

The data entry for the Arrays is completed by a separate function because of how VB handles the dynamic creation of arrays. In particular, it is significantly faster to build the whole array structure first and then fill it with database information, rather than creating the data structure and filling it from the database in a piecemeal fashion.

The *FillArrays()* function is finished once it has created the OD graphical objects as described in Section III.D.

The following snippet of VB code shows the call to four VEGA OD GUI procedures to draw a Bus object, a Line object, a Transformer object and a TransLine object, respectively. The first parameter, "addFlow," in any of the four calls represents an object of type AddFlow ActiveX (indicating where the OD objects are going to be drawn). The second parameter is object-dependent, and specifies which one of our four OD objects is going to be drawn.

```
DesignBus(addFlow, busData)
DesignLine(addFlow, lineData)
DesignTransformer(addFlow, transData)
DesignTransLine(addFlow, transLineData)
```

When applied to all the necessary elements of our problem, the result of the aforementioned steps is the OD representation of the electrical network in "Edit mode." When interdiction results (produced by executing the optimization module) are present, the user is allowed to switch to "Run mode" to visualize them. A detailed description of the functionalities of each of these modes can be found in Chapter IV. Next, we describe the design of these two modes within the OD interface architecture.

2.   Edit Mode

In Edit mode, the user interacts with the on-screen representation of ODs through the OD itself and through two VB forms: *frmBusInfo* and *frmLineInfo*.

The OD allows moving all the displayed objects whose position coordinates are updated automatically.

The *frmBusInfo* form appears when a Bus object is *double-clicked* and the *frmLineInfo* shows up when either a Line or a Transformer object is double-clicked. Both VB forms appear as pop-up windows providing information about the specific object. In addition, these dialogue windows allow the user to modify certain object data. If the user makes changes, there are three "Repaint functions," one for each object, responsible for updating the OD graphical representation. Those functions also perform the redesign of the diagram between Edit mode and Run mode switches.
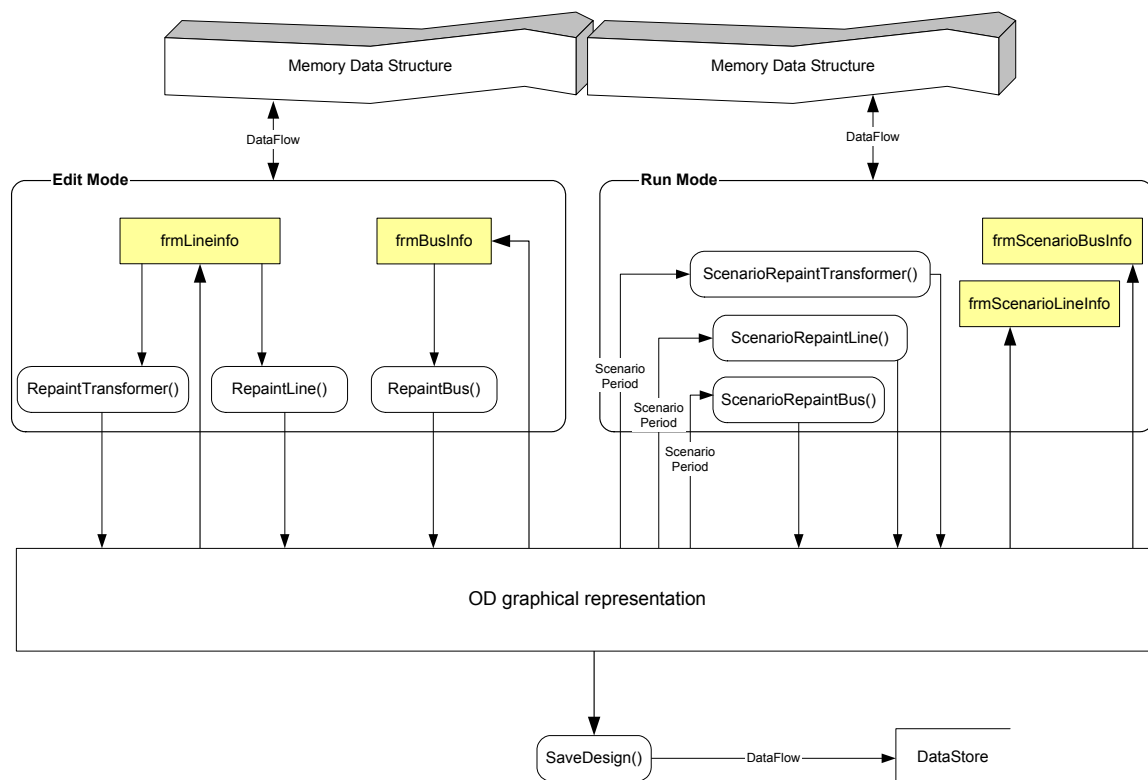
Figure 18.   OD Interface Architecture. In Edit mode the user may view and change data through two VB forms: frmLineInfo, frmBusInfo. Changing data, in turn, may require a "refresh" of part of the OD. This is accomplished by means of three "Repaint functions." In Run mode, data and results can be consulted but not modified. Therefore, the two VB forms (frmScenarioLineInfo and frmScenarioBusInfo) do not require any re-painting action. On the other hand, there are three "ScenarioRepaint functions" used to refresh the OD when the user selects a new scenario or period. In both Run and Edit modes, the information is loaded from the memory data structure (not from the DB).

40

3. Run Mode

As opposed to Edit mode, the user cannot reshape or move the OD objects in Run mode; objects are displayed as positioned in Edit mode. This is done in order to maintain consistency between the existing criteria for editing data and results: If we consider the OD object's coordinates as part of the "system data," it is natural that we can "edit" the OD and save it in the same way that we can open a diagram window (see Chapter IV), to change and save problem data. Run mode is meant to display results, which of course, cannot be modified. Although OD object repositioning does not affect any result, we prefer requiring the user to make modifications in Edit mode only.

In Run mode, the user interacts with the on-screen OD by selecting a particular scenario and time period whose results are going to be displayed. That selection causes changes in the object properties such as label text, colors, visibility, etc. The *ScenarioRepaint* functions refresh the OD diagram to account for changes in the Scenario and Period selected by the user.

As in Edit mode, the user also interacts with the OD through two VB forms: The frmScenarioBusInfo and frmScenarioLineInfo. The two forms provide the user with information and results associated with the selected object. However, unlike the Edit mode dialogue forms, they do not permit changes.

4. Database Update

The SaveDesign() procedure exports all the data that the user is permitted to modify during Edit mode. It exports the data structure from memory to the VEGA system DB.

F. AUTOMATED LAYOUT OF THE INITIAL NETWORK

1. General Considerations

The automated generation and layout of the initial OD helps the user prepare an OD representation with modest effort. The ideas behind the

automated layout of the initial network are essentially the same as those existing in the OD GUI in VEGA 1.0. After the automated layout is completed, the user may modify it as described in Chapter IV Section E.

The function *NewCoordinates()* inside the *DrawInitial* VB module generates the initial network. After the automated layout is completed and with just a few manual changes, the user can plot a nicely fit diagram on the screen that can be saved and retrieved at any time.

In what follows, we describe how buses, lines and transformers (i.e., the objects representing them) are automatically plotted on the screen upon creation of an OD.

2.   Bus Objects

The first step in automated layout is the positioning of Bus objects. In order to find appropriate bus locations, we consider the following parameters:

- Bus X-Y coordinates (provided by the user).

- Initial screen size of AddFlow object in screen coordinates (twips). The size depends on screen resolution.

- Total number of Bus objects in the diagram (case-dependent).

- The desired Bus density (number of visible Buses on the screen at 100% zoom), modifiable by the user (default = 20 buses).

Using these values, the *NewCoordinates* procedure calculates the screen limits of the diagram (in screen coordinates) and positions the Buses according to the new screen coordinates. The initial orientation for all the Buses is horizontal.

3.   Load, Generator and Label Nodes within the Bus Object

The second step is Load and Generator positioning inside a Bus object. We divide the area covered by the OD into four regions: Upper left, upper right,

bottom left and bottom right. Depending on the initial position of each bus (calculated in the previous *NewCoordinates*() step), Load and Generator objects are positioned as shown in Figure 19.

For example, if a bus is located in the upper right region of the OD, we expect more lines (coming into and going out of the bus) directed towards the bottom-left area from the current bus position. Thus, by default, we position the Load object facing up on the right-hand side of the bus, as well as the bus label on the right-hand side of the bus. If generators exist (we expect significantly fewer buses to have generation than to have load), we represent them on the right-hand side of the bus, facing down in order not to conflict with the load, if it exists.
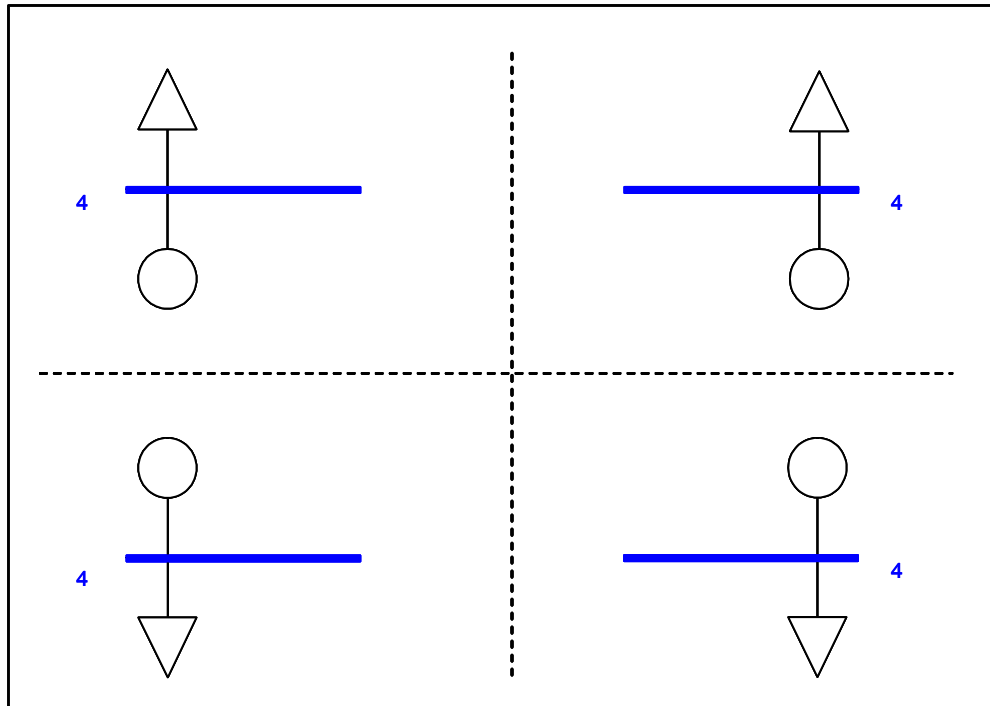
Figure 19. Generator, Load and Label Nodes Automated Positioning. The positioning of the BusGen, BusLoad and BusLabel objects around the BusLine object depends on the relative Bus X-Y coordinates in the OD. Since we expect to see lines leaving from the bus in the direction of the OD center, we attempt to avoid overlapping by positioning objects as in the four examples in the figure: in each example, Bus "4" is located at a different corner of the OD diagram.

As mentioned earlier, in order not to overload the display with extensive details, all load at a bus is represented using a single Load object, even if it is associated with different customer sectors, and all the generation connected to the bus is aggregated too, even if it comes from several generating units.

4.   Line Objects

The third step in automated layout is the design of all Line objects. To enhance the graphical appearance of the multiple lines in the diagram, we position each Line object based on the relative Left and Top positioning of the two buses it connects. For example, in Figure 20, a Bus object instantiated as Bus1 (identified by a BusLine Node and a BusLabel with *text* value "1" in the Figure) is located to the right of Bus3. We establish this relative location by comparing the *.Left* property of both buses. Because *Bus1.Left > Bus3.Left*, we draw the Line L-1-2 beginning closer to the left-end of the BusLine1 Node, and ending closer to the right-end of BusLine3 Node. For the same reason, Line L-1-4 connecting Bus1 and Bus2 has the opposite behavior, as shown in Figure 20. (Remark: BusLine Nodes are displayed horizontally in the automated layout, and they cannot be rotated vertically by the user until this process is completed.)
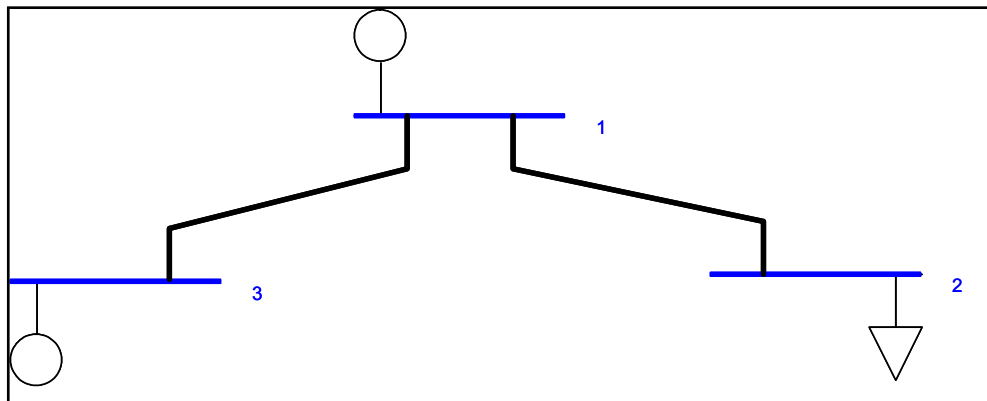
Figure 20.   Line Object Automated Positioning. In order to reduce Line overlap, every Line is positioned depending on the relative location of the two Buses it connects. For example, the first segment of the line connecting Bus "1" and Bus "2" is directed downwards from Bus "1" because Bus "2" is below Bus "1." In addition, the Line starts on the right-hand side of Bus "1" and ends on the left-hand side of Bus "2" because Bus "2" is located to the right from Bus "1."

Similarly, depending on the vertical coordinates of two connected buses, the first and the third segments of the Line object connecting them are positioned accordingly: For example, in Figure 20, Bus1 is located above Bus3 (*Bus*1.*Top* < *Bus*3.*Top*). Therefore, the first segment of the line (anchored to Bus1) is directed downwards from the BusLine1 Node, whereas the third segment (anchored to Bus3) is directed upwards from the BusLine3 Node. Initially, the first and the last segments will always have the same length. The second (middle) segment of the line has variable length depending on the two other segments.

In case that the difference in vertical coordinates between the two Buses is small, the Line will be designed as shown in Figure 21: In case (a) the bus's position is in the upper-half of the OD, so we direct the Line object inward (i.e., towards the center of the OD) because we believe it creates a nicer visual effect, especially if the Buses are actually positioned at the very top part of the OD. In case (b) the bus's position is in the lower-half of the OD, and we direct the line upward.
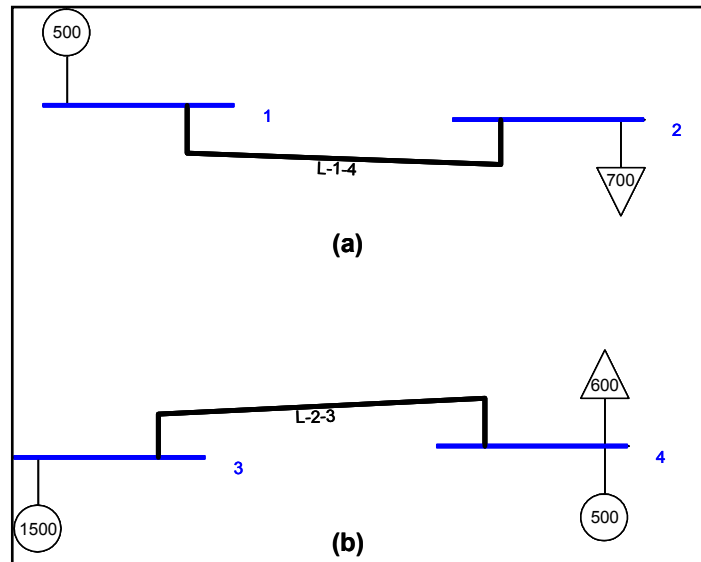
Figure 21. Special Cases of Line Object Automated Positioning. In case (a) the bus's position is in the upper-half of the OD, so we direct the Line object inward. In case (b) the bus's position is in the lower-half of the OD, so we direct the line upward.

45

5. Transformer Objects

The last step is the automated Transformer positioning. Initially, a transformer is always positioned halfway between the two buses it connects. After positioning the main object (Transformer object), the TransLines are created in such way that they always connect the middle of the BusLine and the upper or lower Transformer edge. The decision as to which of the two Transformer edges to pick depends on the relative position between the Bus and the Transformer (Figure 22).
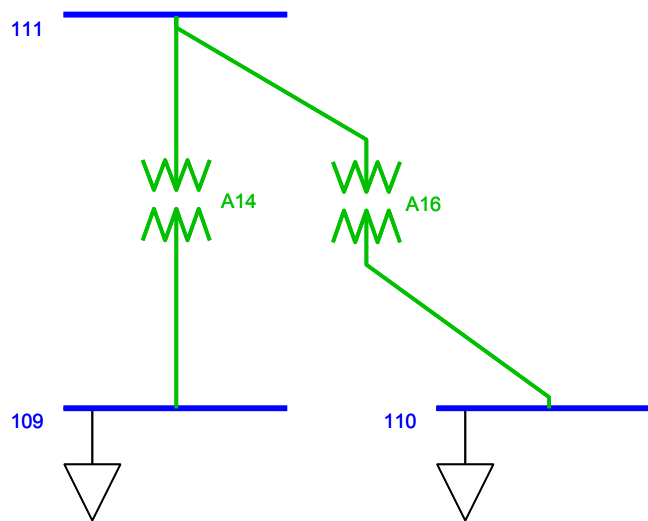
Figure 22. Transformer Object Automated Positioning. The location of the Transformer is always the midpoint between the two Bus objects. With automated positioning, Buses are always displayed horizontally, and Transformers are displayed vertically. At a later stage, both can be rotated by the user.

The location of the Transformer is always the midpoint between the two Bus objects. With automated positioning, Buses are always displayed horizontally, and Transformers are displayed vertically. At a later stage, both can be rotated by the user.

The TransLines are created during runtime and their associated data are stored in a separate table called LineTrans which was not part of the original DB

in VEGA 1.0.   The LineTrans table is the only new table incorporated into the VEGA DB, and it is not used elsewhere in the VEGA GUI.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ONE-LINE DIAGRAM GUI

This chapter explains how the new OD GUI is organized, how it works and its functionalities. It is intended for VEGA 2.0 users and, to a lesser extent, to VB programmers interested in technical details of the most important programming features that make the OD GUI possible. For the latter, paragraphs containing VB implementation details can be recognized as those that begin with a standard VB icon ![VB icon].

A. OVERVIEW

The user can enter the OD GUI through the main menu of the VEGA initial screen or, alternatively, by clicking on the OD button in that screen (Figure 23).
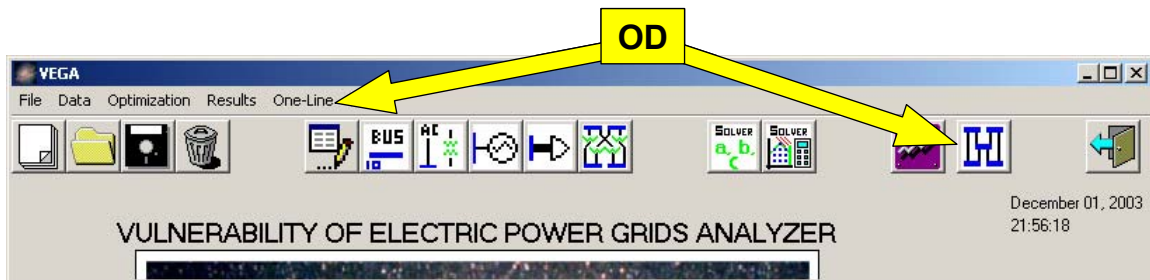
Figure 23.   Access to the OD GUI from the VEGA system.

We assume that, before opening the OD GUI, the following data have been entered into the VEGA system (and are therefore available from the system DB):

- Buses: System buses and associated data.

- Lines: System lines and associated data.

- Transformers: System transformers (if any) and associated data.

- Generators: System generators and associated data.

- Loads: System load per customer sector and associated data.

- Substations: System substations (if any) and associated data.

49

This can be accomplished through the data menu in the VEGA GUI.

The above bulleted items become the basic components of the electric network that we need to represent as an OD. In order to carry out this task, we still require another piece of information: Bus X-Y coordinates. These can be entered manually or imported from an external flat file in CSV (comma-separated value) format. Both the manual and the import procedures can be accessed either from the VEGA main menu or from the OD menu.

When both system data and bus coordinates are available, the "Empty Mode" of the OD GUI allows the user to create a new OD diagram in automated mode, or to import an existing OD for the case. The OD can be edited in "Edit mode." In addition, when "Results" are available (i.e., if the user has carried out interdiction analysis for the case by using the optimization tools within VEGA), the user is entitled to switch to "Run Mode" in the OD GUI and visualize the results graphically. These three working modes, are described in detail later in this chapter.

B.   THE ONE-LINE DIAGRAM GUI SCREEN
1.   Overview

MS-Windows screen settings are assumed to be set to at least 1024×768 pixels of resolution (1280×1024 or higher recommended), and a minimum of 256-color quality. Under inferior quality conditions, some objects may appear blurry or even disappear from the screen.

The OD GUI screen (Figure 24) is displayed in an independent VB window divided into three areas: The **toolbar** area, the **scenario display** area and the **one-line diagram** area. Each of these areas has a different content depending on the selected functionality mode ("Empty", "Edit" or "Run") and depending on the user selection within these modes (e.g., print preview, dialogue box display, etc.)
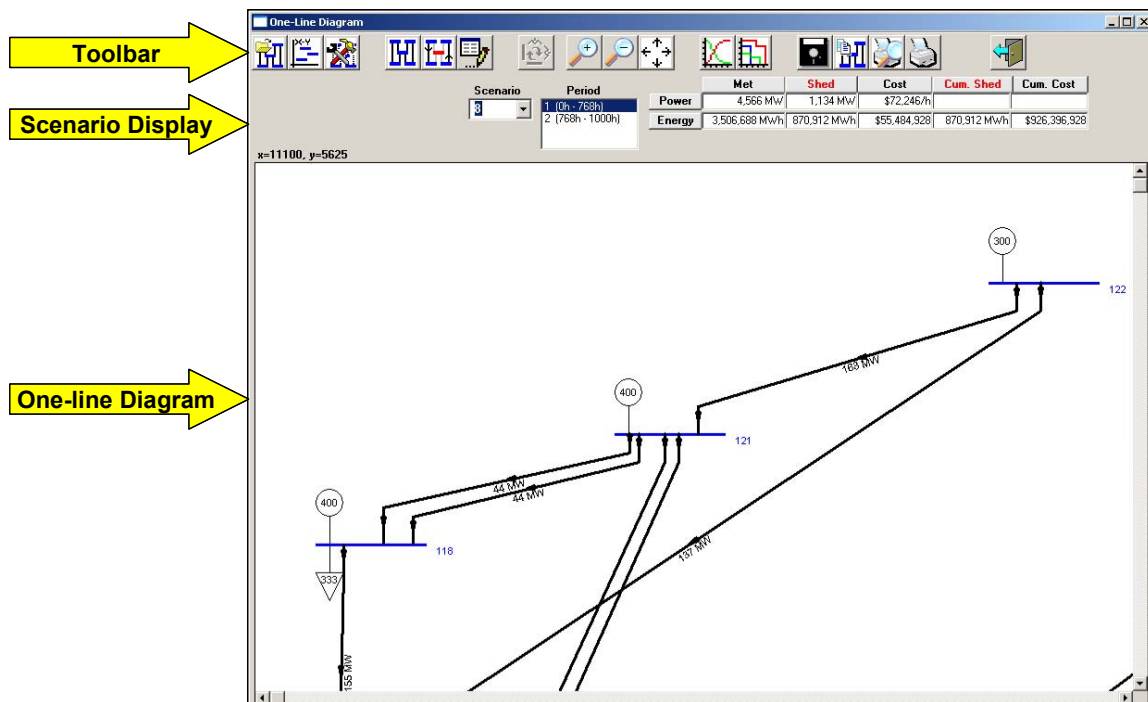
Figure 24.   OD GUI: Screen areas.

📋 The associated VB Forms that contain all the objects accessible from the OD GUI are:

- *Graphical_Design* (main OD GUI manager)

- *frmBusInfo* (dialog box for bus data in Edit mode)

- *frmScenarioBusInfo* (dialog box for bus results in Run mode)

- *frmLineInfo* (dialog box for line and transformer data in Edit mode)

- *frmScenarioLineInfo* (dialog box for line and transformer results in Run mode)

- *Design_Buses_Coor* (dialog box for bus X-Y coordinates, also available from the Bus Data menu of the VEGA GUI)

2.   Toolbar Area

The toolbar (Figure 25) is always docked horizontally at the upper-most region of the OD window.   It contains a total of 17 functional icon-buttons grouped into seven blocks by the kind of task they accomplish.   Groups are separated by empty blocks, called "separators," which allow partial toolbar customization.
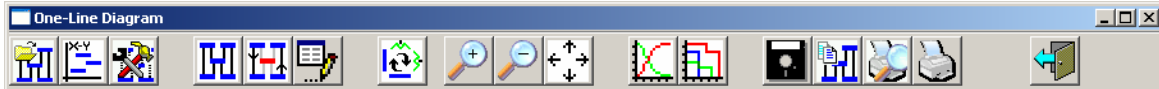
Figure 25.   Toolbar Area. This contains a total of 17 functional icon-buttons grouped into seven blocks by the kind of task they accomplish.

Some of the buttons in the toolbar are disabled depending on the working mode and the current action that is being undertaken.   For example, the "Save" button cannot be used until an OD has been built or loaded.

As the user moves the mouse pointer over any of the enabled toolbar buttons, a description of the functionality is displayed as "tool-tip text."   The action is not carried out until the user clicks on the button and releases the mouse button without leaving the button area.

The functionality of each button is described in detail later in this chapter.

3.   Scenario Display Area

The Scenario Display Area (Figure 26) is located immediately below the toolbar, above the one-line diagram area.   In Edit mode, this area is empty except for the display of X-Y coordinates of the mouse pointer on the bottom-left corner. (See "Options" to switch between screen coordinates and original X-Y coordinates.)

In Run Mode, the Scenario Display Area is filled with overall results for a selected scenario and time period. The scenario is chosen by the user from a drop-down list.   The default scenario selection is "0," which always exists because it is created internally by VEGA. Once a scenario has been selected, a

subset of "periods of restoration" associated with the scenario is shown in a list, from which the user must select a specific period. Recall that these periods after interdiction refer to each stage that the power system undergoes from the attack until all system components are repaired. Typically, we expect lines to be repaired quickly, whereas high voltage transformers and generating units take much longer. The outage duration data are provided by the user for each specific component of the system." Periods are listed using an index followed by a period time-window of the form "*Initial hour – Final hour,*" which is scenario-dependent. The default selection is the first period on the list. In Figure 26, the selected scenario ("16") has four periods, from which the user has selected period "2" covering a time-window between 72 and 360 hours after the attack.
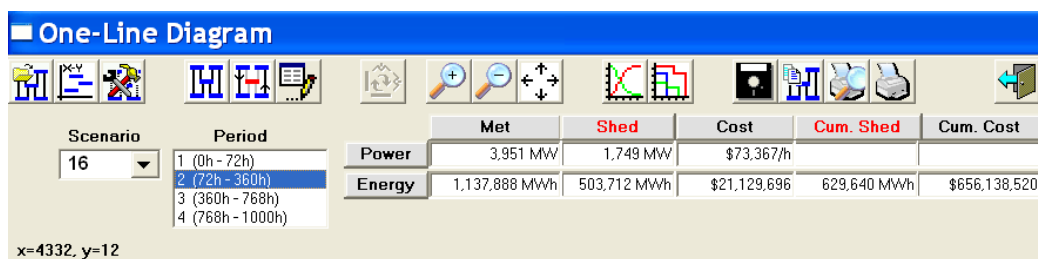
Figure 26.   Scenario Display Area in Run Mode. This consists of two list boxes and a data table. The Scenario list is static, containing Scenario "0" (default scenario created by the system) and all the other Scenarios under analysis (created by the user). The Period table is dynamic, and is refreshed when the user selects a different Scenario. The data table is also dynamic, and is refreshed when the user selects a new Scenario and/or Period.

Once the scenario and period have been selected, the text boxes on the right-hand side of the display area show information about the system status for that period of time: Instantaneous power met and shed, and total cost. These data are also displayed in terms of energy met and shed and energy cost for the whole period. Cumulative energy shedding and cumulative cost from hour "0" to the last hour of the selected period are also displayed.

Selecting a scenario and/or period causes the OD to be "refreshed" according to the results for that particular selection. This causes a nice visual effect, as the user can compare how the electric grid evolves over the periods of

53

restoration (for a particular scenario) by simply moving down on the period list. Also, he or she may compare different ODs representing the first period (which contains the system status immediately-after the attack) for a sequence of scenarios, by simply moving down on the scenario list. (Recall that, by default, period "1" is selected automatically when the user chooses an scenario.)

Mouse coordinates are displayed at the bottom of the Scenario Display Area as the mouse moves over the OD display. Coordinate units are set in the Options menu from the Toolbar.

The associated code for pinpointing X-Y coordinates is controlled under the MouseMove event of the AddFlow object. This means that whenever the mouse is moving over the diagram, the MouseMove event will continually update the value of the Coordinates label (lblMouse) where the coordinates are displayed. Here is an excerpt of the code under the MouseMove event:

```
log_X = (x * 100 / AddFlow1.xZoom) + AddFlow1.xScroll
log_Y = (y * 100 / AddFlow1.yZoom) + AddFlow1.yScroll
lblMouse = "x=" & log_X & ", y=" & log_Y
```

Notice that, in order to provide accurate coordinates, the formula to calculate the coordinates (in OD screen units, or twips) considers the diagram zooming factor and any previously scrolling of the viewable window.

4. One-Line Diagram Area

The OD area (Figure 27) is located below the Scenario Display area, and includes horizontal and vertical scroll bars. It is used to display and edit the OD for the electric system under study, and to preview it before printing.

In Edit mode, the OD is editable in many ways (moving objects, reshaping lines, etc.), and represents system data. In Run mode, the graphic area is not editable. On the other hand, since interdiction, generation, power flows and other results change by scenario and period, the OD display adapts dynamically to accommodate these results. This adaptation produces an insightful view of the

electrical system's robustness or lack thereof, as a function interdiction resource, as well how the system's functionality is restored over time, after interdiction.
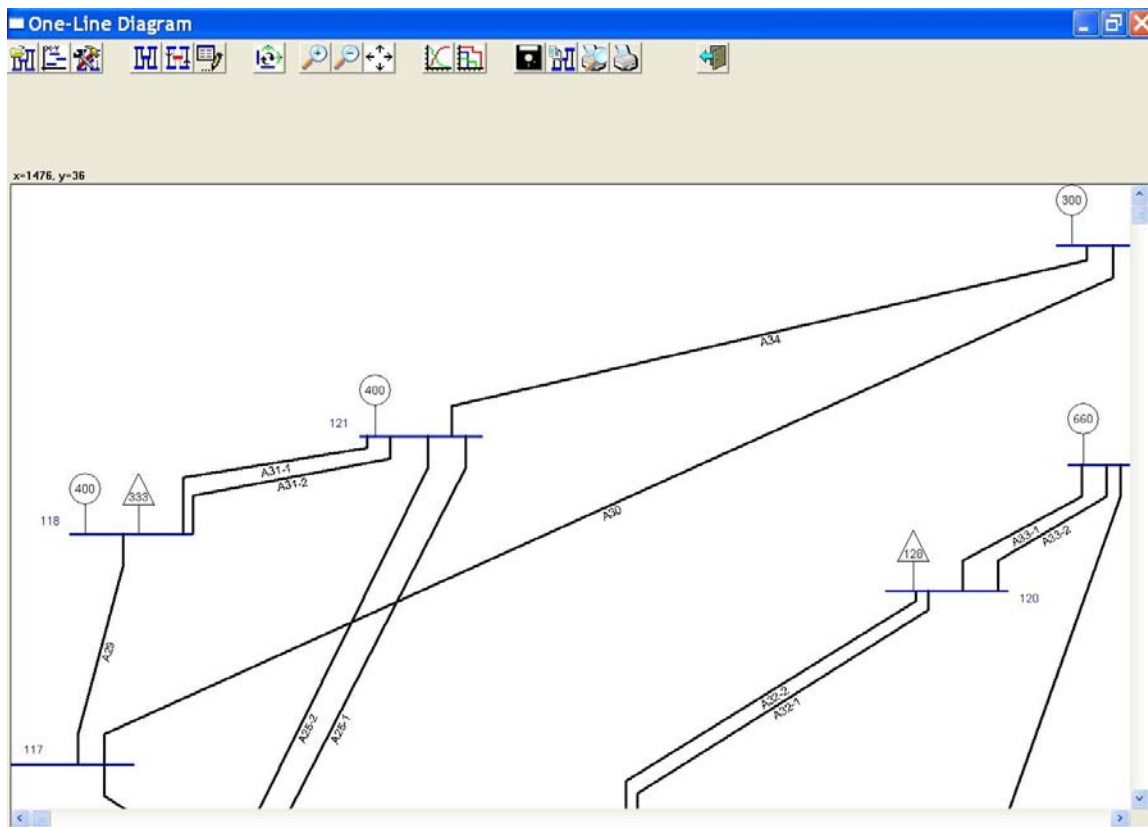
Figure 27.   One-Line Diagram Area. The OD area is located below the Scenario Display area, and includes horizontal and vertical scroll bars. It is used to display the actual OD for the electric system under study. This particular OD is only partial because the user has zoomed in to focus on a particular section of the network.

 Programmatically, the OD area is implemented as an AddFlow object that contains many instances of all the objects described in Chapter III.

C.   THE TOOLBAR

1.   Buttons

In this section we describe the functionality of all the toolbar buttons. Certain buttons are not available in some of the working modes.

Open:

Availability: Empty and Edit modes.

Description: Opens an existing one-line diagram for the case.

Note: In Edit mode, opening an OD brings up the last saved OD in the DB for this case, overwriting the existing OD.

During Empty mode, the button is enabled only if an OD exists in the DB for the case.

Coordinates:

Availability: Empty and Edit modes.

Description: Allows the user to enter or import bus coordinates, and select which buses are to be displayed or hidden.

Changes will not be reflected until a new OD is laid out in automated mode using the "Build" button.

Build:

Availability: Empty and Edit modes.

Description: Automatically lays out a new (tentative) OD from scratch. The building process determines screen coordinates for all objects in the OD, as described in Chapter III Section F.

Edit:

Availability: Run mode.

Description: Edits the OD for the case, and allows selected data and the OD display layout to be modified. Edit mode does not display any problem results.

Run:

Availability: Edit mode.

Description: Displays the OD for the case and incorporates power flows, interdicted components and other system data and results.

The availability of Run is contingent upon the existence of results, which are provided by VEGA's optimization module.

Options:

Availability: Edit and Run modes.

Description: Opens an option window (see Figure 28) from which the user can set a number of default options that affects the OD display. Options may apply to Edit mode, Run mode or both.

Remark: At the conclusion of this thesis project, only the "Use line width" option (among the options listed below) is fully implemented. All the other options have been set to default values; it is assumed they will be made functional in updated versions of the OD GUI.

The available options are:

- Bus size: Sets the bus object size (in twips). (Default=1800)

- Number of buses: Sets the maximum number of buses per full screen at 100% zoom level, after using the automated layout procedure. This

option allows controlling the density of buses in the OD layout. (Default=20 buses.)

- Zoom step: Sets the zoom in/out factor in the range [1.01, 2.00]. A larger value causes a deeper effect every time the user zooms in or out on the OD. (Default=1.25)

- Line code/flow: Sets whether line code labels (in Edit mode) and line power-flow values (in Run mode) are displayed. (Default=Yes)

- Transformer code/flow: Sets whether transformer code labels (in Edit mode) and transformer power-flow values (in Run mode) are displayed. (Default=Yes)
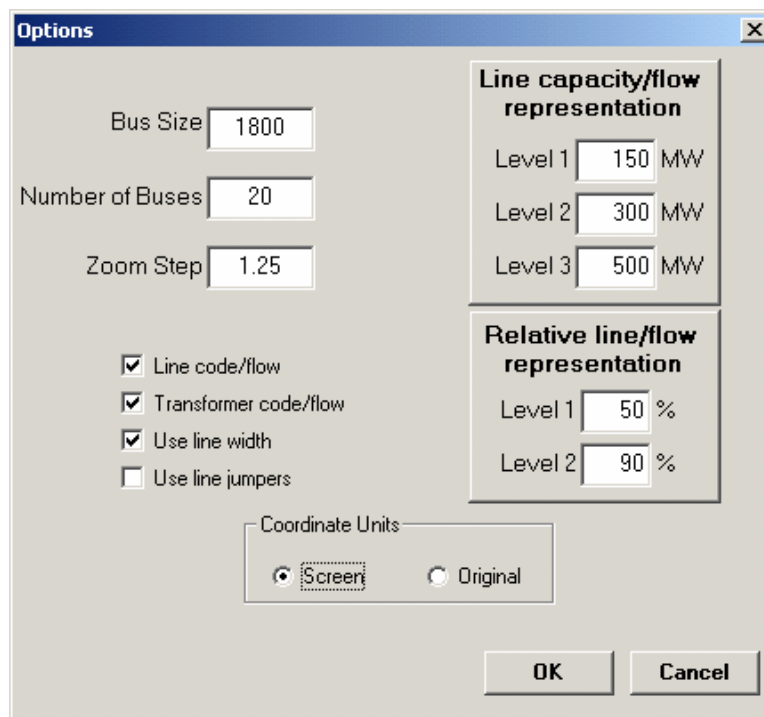
Figure 28.   Options Window. Allows the user to change the default settings for displaying ODs. For example, if we "uncheck" the "line code/flow" option, neither Line codes nor Line flow values will be displayed in Edit and Run mode, respectively.

- Use line width: Only in Edit mode, sets whether lines with higher capacity are depicted using thicker lines or not (i.e., same width for all lines).   (See Line capacity/flow representation option below.) (Default=Yes)

58

- Use line jumpers: Sets whether crossing lines will be displayed using a jumper instead or a straight cross over. (Default=No)

- Coordinate units: Sets the coordinate system for the mouse coordinates as it moves over the OD display. If set to "Original," it uses coordinates based on the original coordinates entered by the user. If set to "Screen," it uses internal OD screen coordinates (twips). (Default=Screen)

- Line capacity/flow representation: In Edit mode, selecting "Use line width" will display lines using heavier line width for larger capacities. The user can establish the "setpoints" (levels in MW) to accomplish this. In Figure 28 for example, lines between 0 and 150 MW will be displayed lighter than lines between 150 and 300 MW. In Run mode, the same values are used to represent actual power flows.

- Relative line flow representation: This option is used only in Run mode. It sets relative setpoints (% of line capacity) for the arrow size used to represent power flow direction. Three arrow sizes exist: Small for power flows under the first level mark, medium for power flows between the first and second marks, and large for power flows over the second level mark. For example, in Figure 28, lines carrying power flows over 90% of the maximum line capacity will be displayed using a larger arrow than lines carrying power flows between 50% and 90% of that capacity.

Rotate:

Availability: Edit mode.

Description: Allows the user rotate a selected bus or transformer from horizontal to vertical display or vice-versa. The selected Bus or Transformer object in the OD is the one that has been *clicked* prior to the

rotate button. Lines, generators, loads and labels associated with the rotated object are rotated and/or relocated accordingly.

The code associated with this feature can be found in the DrawDesign module and consists of two functions, RotateBus and RotateTransformer.

Zoom-In:

Availability: Edit and Run modes.

Description: Zooms in the current OD. This makes the display of the OD larger. It closes on the upper left corner of the current screen. The zoom factor is set in the options.

The code associated with Zoom In makes use of the built-in properties of AddFlow, namely xZoom and yZoom. By increasing or decreasing those two values, the size of all the objects inside the AddFlow object changes accordingly. For example, the code:

```
AddFlow1.xZoom = AddFlow1.xZoom * 1.25
AddFlow1.yZoom = AddFlow1.yZoom * 1.25
```

will cause a 25% enlargement in the current display. An overall upper limit of 500% has been set to prevent potential overflow errors.

Zoom-Out:

Availability: Edit and Run modes.

Description: Zooms out the current OD. This makes the display of OD smaller. It moves away from the upper left corner of the current screen. The zoom factor is set in the options.

 This feature is coded like the above Zoom-In feature. The code:

```
AddFlow1.xZoom = AddFlow1.xZoom / 1.25
AddFlow1.yZoom = AddFlow1.yZoom / 1.25
```

will cause a 25% decrease in the display area. An overall lower limit of 20% has been set to prevent potential overflow errors

 Isofit:

Availability: Edit and Run modes.

Description: Adjusts the zoom automatically in order to make the entire OD fit in the screen (Figure 29). For very large and/or unevenly distributed networks the size of objects displayed in the OD will be too small for practical analysis.
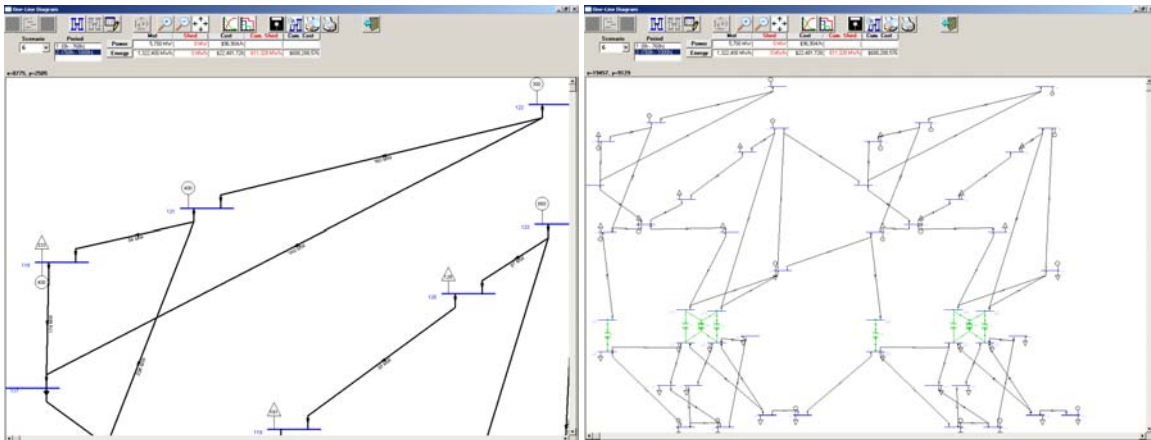
Figure 29.　　An OD Before (left) and After (right) Using Isofit.

 The OD is resized so it can be viewed inside the viewable part of the AddFlow window. The code associated with this feature calculates the xZoom and yZoom properties so that the diagram fits in the AddFlow window.

 Graph1:

61

Availability: Run mode.

Description: Displays a graphic with a summary of interdiction results by scenario. This graphic is the same as the one available from the Result Menu in the VEGA GUI.

Graph2:

Availability: Run mode.

Description: Displays a graphic with a summary of interdiction results by scenario and period. This graphic is the same as the one available from the Result Menu in the VEGA GUI.

Save:

Availability: Edit mode.

Description: Saves changes made to the OD display and data.

> Any changes the user makes to the One-Line Diagram while in Edit mode are stored temporarily to the memory data structure described in Chapter III. The code associated with the Save button transfers all the data that exist in the memory data structure to the appropriate fields in the database, overwriting any existing values.

Copy:

Availability: Edit and Run modes.

Description: Exports the current OD to the Clipboard as an enhanced metafile picture.

Remark: The Clipboard object is shared by all Windows applications. Therefore, its contents are subject to change whenever any form of "Copy" is used in VEGA or in any other MS-Windows application.

 The code associated with this button uses the ExportPicture method of AddFlow object and SetData method of VB Clipboard object. The Clipboard object can be used to enable a user to copy, cut, and paste text or graphics in an application. Before copying any material to the Clipboard object, its contents should be cleared by executing the Clear method.

```
Clipboard.Clear

Clipboard.SetData AddFlow1.ExportPicture(afAllItems, False, True)
```

 Preview:

Availability: Edit and Run modes.

Description: Previews the OD as it will be printed. Initially, the OD zoom level is maintained. A *left-click* on the Preview display provokes a zoom in, and a *right-click* provokes a zoom out. The OD may cover more than one page, but only one page can be previewed at a time. The combination *Shift + double-right-click* advances one page, whereas *Shift + double-left-click* goes back one page. The aforementioned combinations are established by the AddFlow ActiveX control.

If already in Preview, the user can return to the OD display (in Edit or Run mode) by *clicking* again on the Preview button or in the Exit button.

 Print:

Availability: Edit and Run modes.

Description: Prints the current OD to the MS-Windows default printer. Only the default printer can be used. To ensure the desired printout, the user should Preview the OD first (see above).

Both Preview and Print use an extension of *AddFlow* called *PrnFlow*. This is also an ActiveX control that allows printing *AddFlow* diagrams. Some of the *PrnFlow* functions are: Multi-page printing, print previewing, margins, header and footer manipulation . Currently, the OD GUI does not take full advantage of all of these features. The functions implemented at this point are PrintPreview and Mutil-page printing.

Exit:

Availability: Empty, Edit and Run modes.

Description: Exits the OD GUI, except when Preview is displayed, in which case it returns to Edit or Run mode.

2.   Customization

The toolbar can be customized by *clicking* on any of the empty areas between buttons. Customization is handled directly by standard MS-Windows procedures: Adding buttons, removing buttons, moving buttons up and down, and resetting the toolbar (Figure 30).
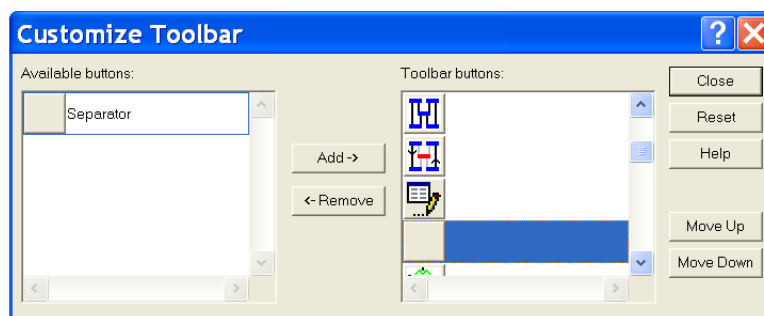

Figure 30.   Toolbar Customization Window.   Buttons can be removed, added or arranged differently by moving them up and down.

D. EMPTY MODE

1. Introduction

The Empty mode occurs when the OD display area is empty.

The toolbar buttons that are enabled in Empty mode are Open, Coordinates, Build and Exit (Figure 31).

Figure 31.   Toolbar in Empty mode. The only buttons that are enabled are Open, Coordinates, Build and Exit.

Paragraphs 2 through 4 below describe the circumstances that can lead to Empty mode in the OD GUI.

2. First Time the OD GUI is Used

The first time a user enters the OD GUI for a new case study, the system will be waiting for the user to *click* on the "Build" button in the Toolbar to layout an OD automatically. This button will be enabled if the user has provided X-Y coordinates for at least one system bus.

The Build button uses the original bus coordinates to automatically create bus, line and transformer screen locations, and to save them into the VEGA DB (see Section III.F). Next, it brings all of the electric system data and available results from the DB into the system memory. Then, network objects are positioned on the screen into the AddFlow object (see Section III.E). The code that controls this process is located in the Private Subroutine Build_oneline() of the VB form *Graphical_Design*.

3. Bus Visibility and X-Y Coordinates

By default, upon creation of a bus entity in the VEGA system DB, the bus is assigned X-Y coordinates equal to (0,0). For this reason, buses are set to "invisible." It is the user's responsibility to provide actual coordinates or set the

65

"visible" property to "true" for at least one bus in order to represent an OD. At that point, the "Build" button will be enabled in order to create a new automated OD.

Coordinates can be entered into the system in a manual or automated procedure through the Bus Coordinate window originally available from the Bus Data form in the main menu of the VEGA system. We have made this form available from the OD GUI (by clicking on the "Coordinates" button) for practical reasons: A user might want to enter all the information pertaining to the bus at once, or perhaps he or she prefers to use the VEGA system without ODs, leaving that part for later. Thus, it seems natural to be able to enter and/or have access to the coordinate manager from both the Bus Data window and the OD GUI.

Figure 32 shows the Bus Coordinates window, where the user may edit the coordinates for each bus, or import all (or part) of them at a time from a standard file in CSV format.

Figure 32.   Bus X-Y Coordinates Window. The user may enter coordinates for some Buses, and/or import them from a CSV file. In addition, the user selects which buses should be visible in the OD. Using the default button causes all buses with (0,0) coordinates to be invisible, and all buses with non-(0,0) coordinates to be visible.

Although it is possible to set which buses are visible or hidden manually (i.e., on a one-by-one basis), the "Default" button at the bottom of this window

greatly simplifies this task: It makes all buses with non-(0,0) coordinates visible, and all buses with (0,0) coordinates invisible. Since (0,0) are the initial coordinates of all buses, the user only has to worry about entering coordinates for the buses he or she desires to make visible. (Perhaps a manual change is needed to make visible a specific bus which is actually located at (0,0), but all the others will be updated automatically.) Hiding buses is especially useful for large cases, for which a proper visualization of the OD would be impossible if all buses were displayed.

Remark: Even if some buses are invisible, their data remains in the system and we can access them through bus-line navigation, as explained later.

The OD GUI directly invokes the coordinate manager form, *Design_Buses_Coordinates*, which handles the optional call to the automated coordinate import form, *Design_Buses_Import_Coordinates*.

4. An OD Already Exists for the Case

If the user enters the OD GUI while a previous OD exists for the case (stored in its DB), the "Open" button will be enabled on the toolbar, and the user may simply click on it in order to open the existing OD. Alternatively, the user may prefer to add or modify bus coordinates and then build a new OD instead of displaying the existing one. Future versions of the OD GUI may use the "Open" button to display different saved ODs for the same case (e.g., showing a different number of buses).

The "Open" button transfers all bus, line and transformer coordinates (which might differ from the original coordinates if the user has made and saved modifications to the initial OD), along with all other associated data and available results, from the DB into the system memory. Then, all network objects are positioned on the screen inside the AddFlow object (see Section III.E). The code that controls this process is located in the Private Subroutine Open_oneline() of the VB form *Graphical_Design*.

E.   EDIT MODE

1.   Introduction

The Edit mode allows the user to view and modify OD displays, along with selected data for the case under study. It may be viewed as the default mode in the OD GUI.

The system enters this mode automatically when a new OD is built or an existing OD is opened. When results are present, the user may switch from Edit mode to Run mode by clicking on the Run mode button. The user may then return to Edit mode using the Edit button.

2.   Toolbar Buttons

The following toolbar buttons are available in Edit mode: Open, Coordinates, Build, Run, Options, Rotate, Zoom-In, Zoom-Out, Isofit, Save, Copy, Preview, Print, Exit (Figure 33).

The Power Flow (Run) button is only enabled when results exist in the system DB.
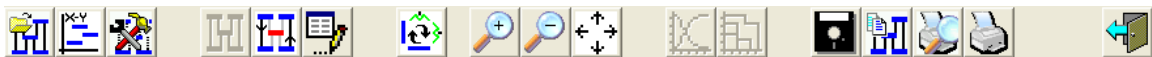
Figure 33.   Toolbar in Edit Mode. The only buttons that are not enabled are Edit, Graph1 and Graph2. The Run mode is available only if there are results for the case.

3.   Working with the OD

a.   OD Components

In Edit mode, the displayed OD consists of system buses, generators, loads, transformers and lines, which we call generically "objects." An OD can be displayed as long as at least one system bus is visible.

Examples of bus, generator and load objects are shown in Figure 34. Buses are represented using a straight blue line with a label that indicates the bus code in the system. Each generator object and each load object are

associated with a unique bus object. Generators are displayed as circles attached to a bus. Each generator object contains a label which, in Edit mode, displays total generating capacity available at the bus. Similarly, every load is drawn as a large triangle connected to the bus. In Edit mode, the inside area of the triangle shows total load at the bus. Generators and loads are physically "anchored" to the bus.
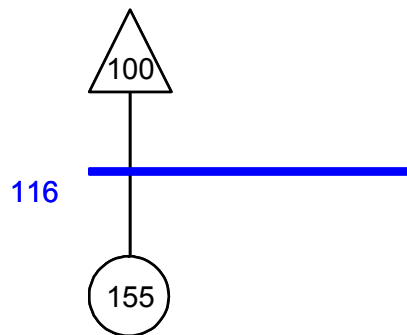
Figure 34. Bus Representation: A bus (and its label) with associated generation and load. Buses in the OD GUI are represented by a straight blue line and a label showing the (unique) bus code. If one or more generators are connected to the bus, they are represented in an aggregated mode by a circle attached to the bus line. Similarly, if one or more loads exist at the bus, they are displayed using a single arrow. The values inside the generator and the load display total generating capacity and total load at the bus, respectively.

System lines (Figure 35, left) are displayed as black, piecewise-straight lines connecting two buses. Our ODs use three segments to represent each line: The first and the last segments are always directed perpendicularly to the bus being connected, whereas the middle segments one connects the other two segments as necessary. Lines connecting buses and transformers are represented similarly (Figure 35, right). Each connection uses a three-segment line. Transformers are displayed in green.
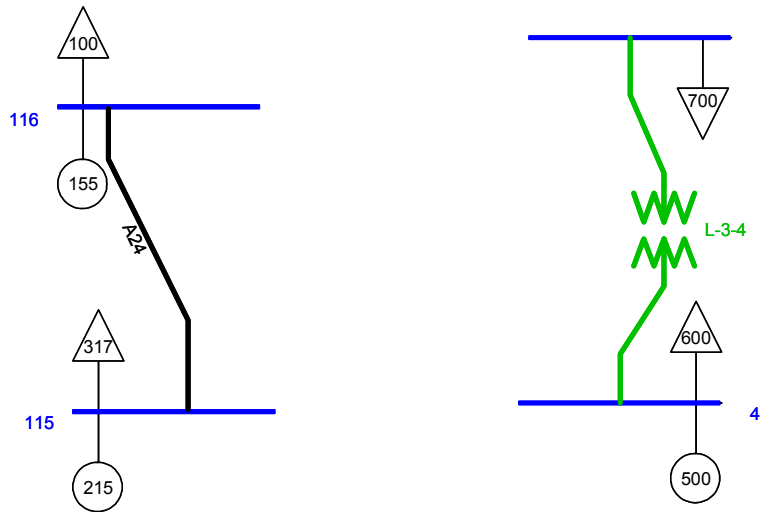
Figure 35.   Line Representation. One line connecting two buses (left) and two lines connecting two buses to a transformer (right).

In Edit mode, the OD has many features that help shape the final diagram for better visualization. Other features allow view information that may be important at a particular point, but is not desired to be seen at all times. Some of the allowed features of OD in Edit mode are:

- Moving buses, generators, loads and transformers.

- Moving and reshaping lines.

- Rotating buses and transformers.

- Using options, tips and dialogue windows to customize the information to be displayed.

b.   *Movement*

Object movement is accomplished by standard MS-Windows drag-and-drop actions using the mouse.

Buses and transformers can be repositioned anywhere inside the OD display area. A modification to the location of a bus or a transformer is followed by an automatic movement of lines, and any generator and load objects connected to the bus (Figure 36). To drag a bus, the mouse click must occur

anywhere in the bus area except where another object (generator, load, line or label) exists.
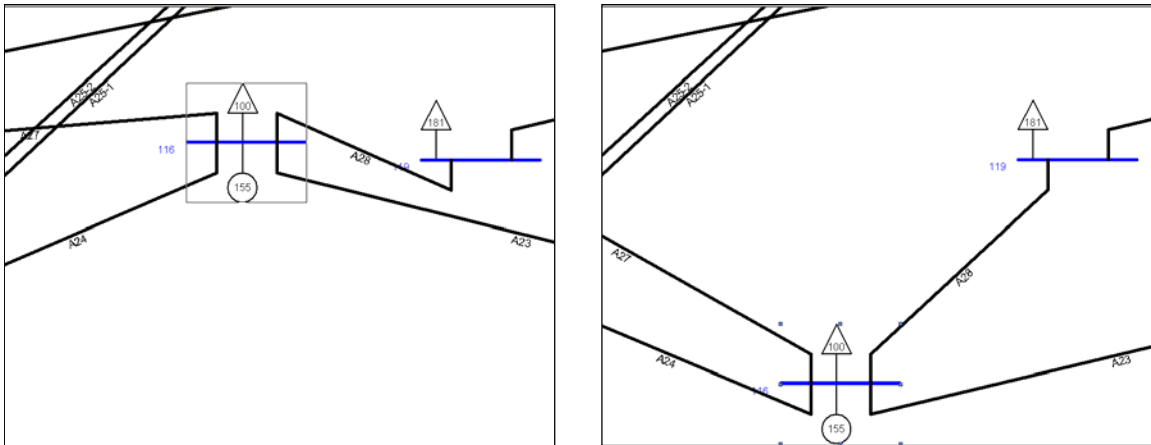
Figure 36.   Bus Movement. The example shows how the user can reposition bus "116," along with all of its associated components (generator, load, label and lines). This is accomplished by the standard MS-Windows operation of "drag and drop." The *click* before start to dragging the bus should occur inside the bus area (imaginary box around the bus, see left panel) but not on any of the other objects inside the bus.

A generator and/or load connected to a bus can also be repositioned anywhere inside the area covered by the bus (Figure 37). To move a generator or load, use the mouse to left click inside the circle or triangle areas, respectively, and (while keeping the left mouse button pressed), move the generator or load to the desired position.
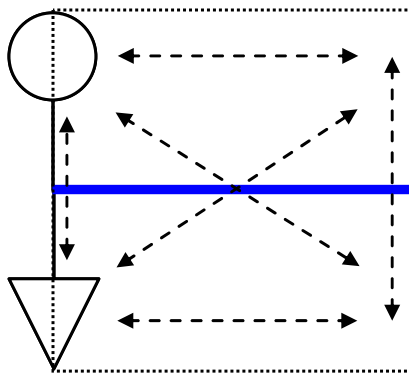
Figure 37.   Load and Generator Movement. The user can drag and drop a load or generator anywhere within the bus area, as shown in the figure.

Line repositioning is available only by using the LinkPoints of each of the line segments. These LinkPoints are movable. The LinkPoints on the buses can be moved along the bus line; the LinkPoints not attached to the buses (on the middle segment) can be moved towards or opposite to the bus line. This is shown in Figure 38.
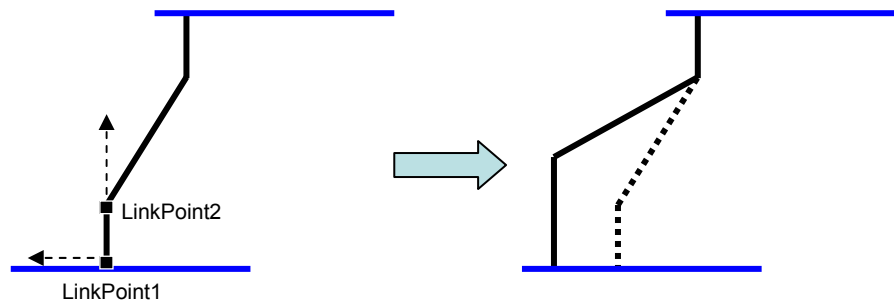
Figure 38.   Line Movement. The line (left) is going to be reshaped in two steps: In the first step, LinkPoint1 (attached to the bus) is shifted left along the bus line. In the second step, LinkPoint2 is moved upwards.   (The order for these steps is interchangeable.) The result is the solid line in the figure on the right, where the dotted line represents the line location before changes, and is shown for reference only.

 The programmatic details of object movement are described in detail in Section D of Chapter III.

Buses and transformers can be rotated from vertical to horizontal, and vice versa, in the OD (Figure 39). This is accomplished by selecting the object to be rotated with the mouse, and then using the Rotate button. Lines follow accordingly, although some manual tuning may be needed.
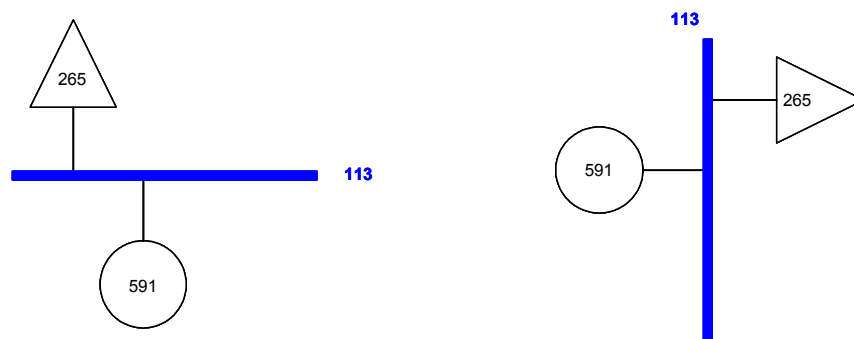
Figure 39.   Bus Rotation. A bus prior to (left) and after (right) rotation. A repeated rotation command returns the bus to its original position.

c. Information Displayed

The information provided by the OD is separated into three levels: Permanent, tip, and advanced.

The "Permanent" level refers to the information that is displayed at all the times by just looking at the diagram (Figure 40):

- Colors and shapes are used to identify the object type.

- Labels inside or around the object show the following information. For buses: Bus code and substation where the bus is located (if any). For generators: Total generating capacity at the bus. For loads: Total load at the bus. For lines and transformers: Code (optional).

- Line width (optional): Depicts higher-capacity lines using thicker lines. Default values for the capacity levels are: Level 1 = [0,150) MW; Level 2 = [150,300) MW; Level 3 = [300,500) MW; Level 4 = [500,∞) MW. The capacity level values are user-defined through Options. The number of capacity levels (four) is non-modifiable.

Line width is established through the DrawWidth property of the AddFlow Link object. The values used for the different levels are Link.DrawWidth = 0, 1, 2 and 3 for capacity levels 1, 2, 3 and 4, respectively.
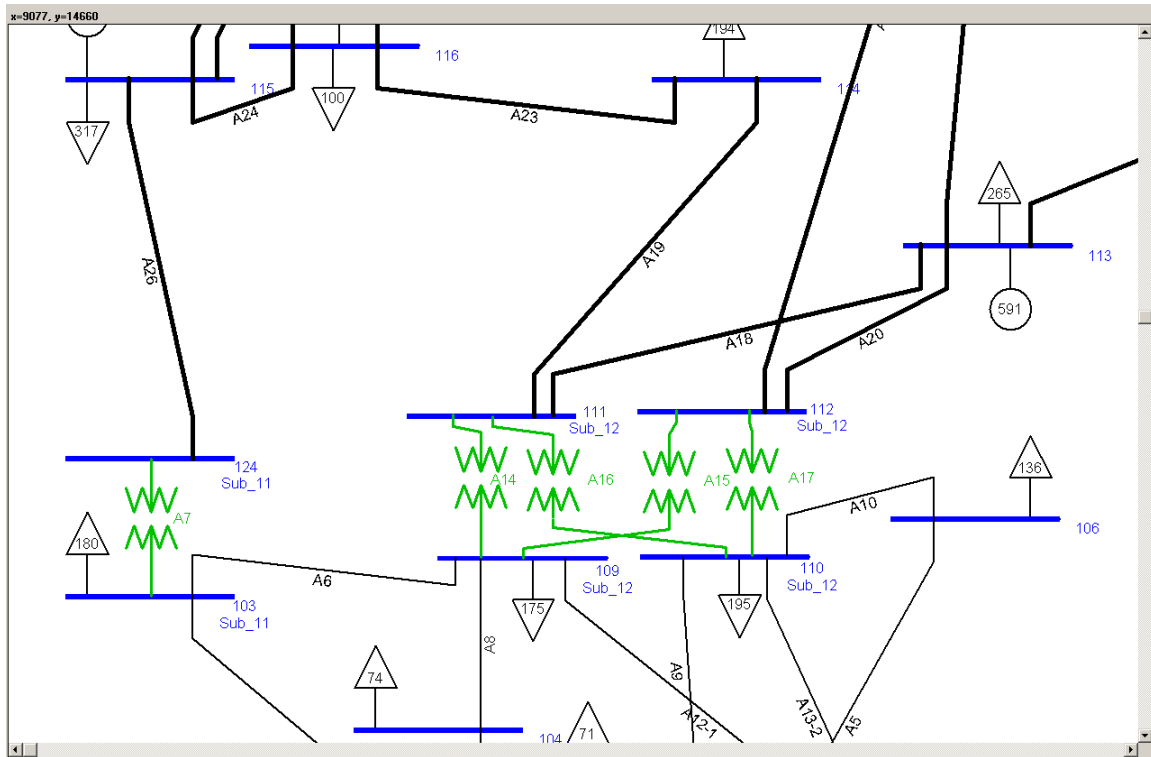
Figure 40.   Permanent Level of Information in Edit Mode.   The OD GUI provides
object type identification using different object shapes and colors. Labels provide object
codes and representative values.

The second level of information is provided by automatic "tips."   In
addition to the information available from the "permanent" level, text tips for
objects are available when the user passes the mouse pointer over an object
(Figure 41).

In Edit mode, the information displayed by these tips is:

- Buses: Code and full name.

- Lines and transformers: Code and capacity.
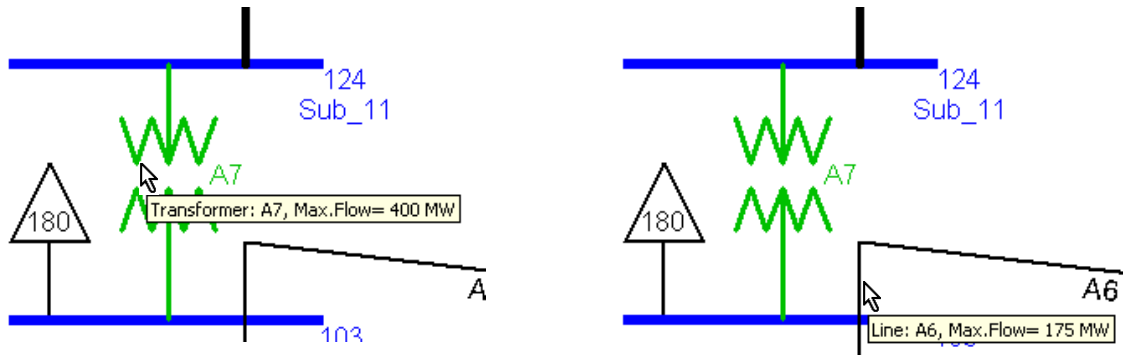
Figure 41.   Tip Level of Information. When the user passes the mouse pointer over an object, additional information on the object is displayed.

The last and most detailed level of information is the advanced level. This information is displayed when the user double-clicks on a bus, line or transformer of the OD. This causes a dialogue associated to the object to open, from which the user can view and modify selected data, and even navigate to dialogue windows pertaining to other objects. This advanced functionality is described in detail in the next two paragraphs.

As mentioned in Chapter III, the OD window is an AddFlow object. This means that any *click* in the OD area is an AddFlow-controlled event. Specifically, the events controlled whenever the user moves the mouse pointer over the AddFlow are:

- *MouseUp* and *MouseDown*: Triggered when the user presses (MouseDown) or releases (MouseUp) a mouse button over the AddFlow object.

- *MouseMove*: Triggered when the user moves the mouse over the AddFlow object.

- *DblClick*: Triggered when the user presses and releases a mouse button (left or right) and then presses and releases it again over the AddFlow object.

Note that, initially, we do not differentiate events by which button of the

mouse is used (right, left or mouse-wheel). This is controlled by VB attributes associated with the mouse event: Button, Shift and vbRightButton. An example of the series of AddFlow events that leads to a *drag and drop* action for an object is:

- *MouseDown*: Object selection and type recognition.

- *MouseMove*: Object control depending on the object type. Movement restrictions are applied.

- *MouseUp*: End of object movement. The new position is saved in the memory data structure.

4.   Bus Dialogue

The Bus Dialogue Window (Figure 42) provides specific information about the selected bus and associated loads, generators, lines and transformers. It is divided into the following sections: Title, Interdiction, Generators, Load Sectors, and Lines/Transformers.

The Title section displays the bus code and name.

The Interdiction section displays and allows user modification of interdiction data for the bus.

The Generator section, displayed in tabular form, provides the information about all generating units at the bus. The user may modify all the displayed fields except the Generator's code. Every time the user changes the "maximum generating capacity" for any generator, the total "maximum generating capacity" for the bus is updated. This, in turn, may affect the graphical representation of the Generator object in the OD. For example, in the extreme case that the user reduces the generating capacity to zero for all the generators at the bus, the generator object will disappear from the OD. If later the user increases again the generating capacity, the generator object will reappear at the same place.

The Load (by Sector) Section behaves similarly. Here, the different sector loads at a bus are displayed and their attributes can be modified.

The Lines/Transformers Section provides tabular information on both incoming and outgoing lines for the selected Bus.   Line codes are shown in black, whereas transformer codes are shown in green.   Unlike the two other tables, this table's fields are not editable.

The user can navigate to the dialogue window of any of the listed lines or transformers directly.   To do this, it suffices to *double-click* on its code.   From there, it is possible to open a dialogue window for any of the two buses connected to the line or transformer (see the following paragraph).
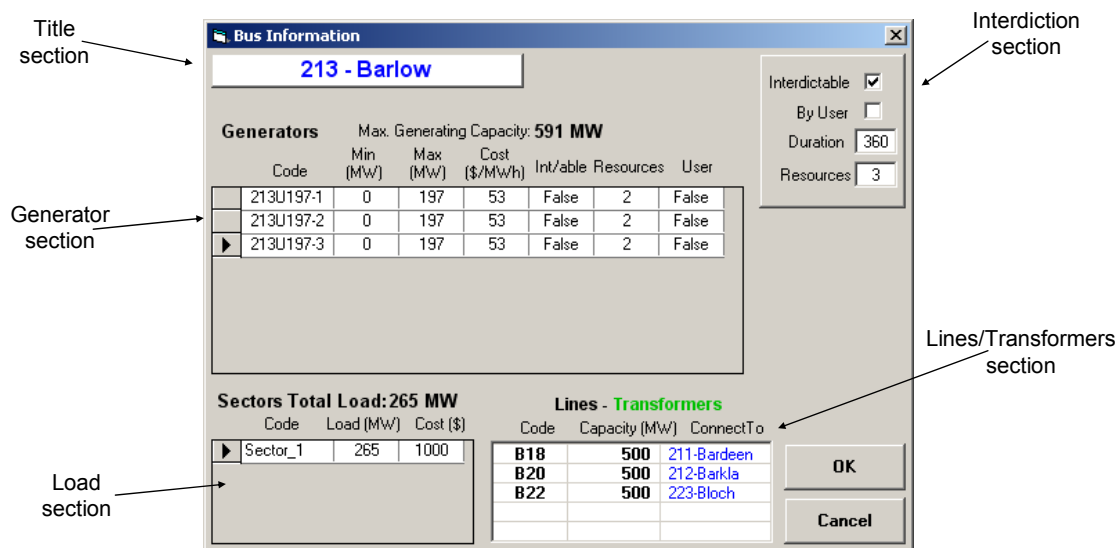
Figure 42.   Bus Dialogue Window in Edit Mode. The bus name and code appear in the title section.   The window shows data for: interdiction parameters for the bus, generating units connected to the bus, loads at the bus, and lines and transformers connected to the bus.   These can be used to open the associated line or transformer dialogue window directly, by simply *double-clicking* on it. The user can modify all values, except codes. For example, the bus can be made non-interdictable with a simple click.

All the information shown in the Bus dialogue window is transferred from the memory data structure. Any changes by the user to the modifiable data, do not store back to the memory data structure until the user clicks on the OK button.   Clicking the OK button updates the memory data structure, along with refreshing the OD. Clicking the Cancel button cancels.

Generator and Sector information is displayed in VB *DataGrid* objects.

Line/Transformer information uses the VB *ListView* object.   This object supports the *DblClick* event.   When the user *double-click*s on any listed line or transformer, the underlying code for the *List View* object unloads the Bus Dialogue window (saving in memory any changes first, as if the OK button had been pressed) and then loads the Line/Transformer dialogue window.

5.   Line/Transformer Dialogue

The Line/Transformer Dialogue Window (Figure 43) provides specific information about the selected line or transformer, along with its associated buses.   It is divided into the following areas: Title, Interdiction, Capacity and Buses.
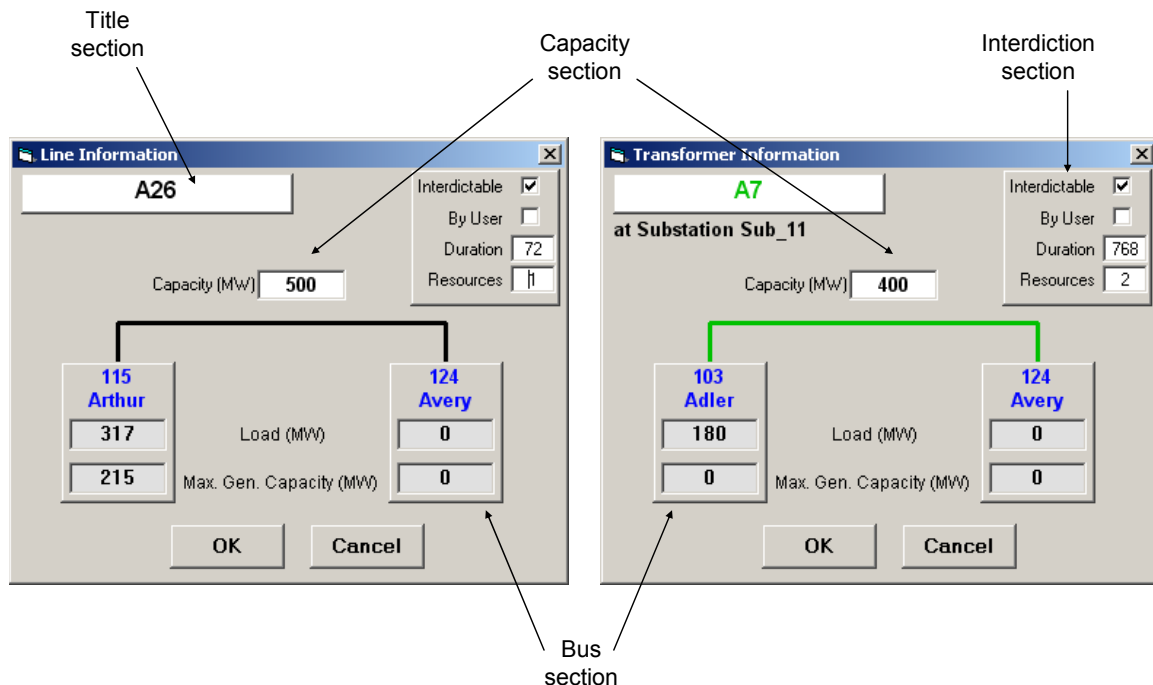
Figure 43.   Line/Transformer Dialogue Window.   The line or transformer code appears in the title section. The window shows data for: interdiction parameters for the line or transformers, capacity, and buses connected to the line or transformer. These can be used to open the associated bus dialogue window directly, by simply *double-clicking* on it. The user can modify all values, except codes and bus data.

The Title section displays the entity's code, in black for a line or in green for a transformer.

78

The Interdiction section displays interdiction data for the Line or Transformer, allowing the user to modify it, such as:

- If the Line or Transformer is interdictable or not.

- The duration of the interdiction in hours along with the resource necessary to interdict this Line or Transformer.

The Capacity section shows the line or transformer capacity in MW, and allows the user to modify this value.

The Bus section shows the two buses associated with the line or transformer, connected by a black or green line, respectively. The information displayed for the buses is not modifiable.

The user can navigate to the dialogue window of either of the two buses by *double-clicking* on the code-name area for the desired bus.

<table>
<tr><td>

The information storage for Line/Transformer is handled as it is in the Bus dialogue window.

The two boxes at each side of the line are VB *Label* objects and support the *DblClick* event. When the user *double-click*s on any listed line or transformer, the underlying code for the *List View* object unloads the Line/Transformer dialogue window (saving in memory any changes first, as if the OK button had been pressed) and then loads the Bus dialogue window.

</td></tr>
</table>

F.   RUN MODE

1.   Introduction

When results exist, the user can analyze them graphically in Run mode. In particular, he or she can:

- Compare attack plans for different scenarios, by simply moving along the elements in the scenario list.

- Analyze system restoration dynamics for a particular scenario, by simply moving along the elements in the period list.

- Analyze the whole system in detail at a given scenario and period, using on-screen graphical information and dialogue windows.

These visual features, in turn, help provide insight into the network's behavior (e.g., robustness, or lack thereof, key components, etc.) and potential level of disruption caused by terrorist attacks. All of this constitutes the ultimate goal of the VEGA OD GUI.

The user may alternate modes of OD GUI between Run and Edit mode by clicking the Run or Edit button respectively. The user may also unload the OD GUI and go back to the VEGA main screen by clicking the Exit button.

In Run mode, a collection of ODs actually exists: For each interdiction-resource scenario, there is an optimal interdiction plan. Each of these plans entail restoration over time, as interdicted components are repaired or replaced over time and load shedding diminishes. This, in turn, causes different system "states" [Salmeron et al. 2003-II], each one characterized by a subset of outaged components and particular system values (power flows, generating unit outputs, bus voltages, etc.), that we represent with different ODs.

Using the OD GUI in Run mode we can view graphically and numerically:

- The consequences of optimal attacks on the network by interdiction-resource scenario, and

- The power flow through the network during the various periods of restoration following an attack.

2. Toolbar Buttons

The following toolbar buttons are available in Run mode: Edit, Options, Zoom-In, Zoom Out, Isofit, Graph1, Graph2, Copy, Preview , Print and Exit.

Figure 44. Toolbar Buttons in Run Mode. In this mode, the Open, Coordinates, Build, Run and Rotate buttons are not enabled.

80

3. Working with the OD

a. *OD Components*

The OD GUI makes use of the scenario display area (already described in Section B, see Figure 26) to determine which OD from the OD collection is going to be displayed. In what follows, we assume that a specific Scenario and Period have been selected from the appropriate lists.

Part of the information we use in Run mode is how system components are affected by the interdiction plan. For each component, this information is divided into "interdiction behavior" and "status," as described below:

The possible (scenario-dependent) interdiction behaviors for each component are:

- Interdicted: The component has been interdicted (e.g., a line has been attacked and destroyed).

- Indirectly Interdicted: The component has not been interdicted but it is connected to a component that has been interdicted, rendering the component inoperative (e.g., a bus is interdicted, which in turn disconnects any generators at the bus, and thereby effectively renders any such generators inoperative).

- Not interdicted: None of the above.

The possible (scenario- and period-dependent) status for each component is:

- In Service: The component can be used (for example, it was not interdicted or, if it was, it has already been repaired).

- Not In Service: The component is inoperative (i.e., either the component or an associated component was interdicted and still needs to be restored).

The OD objects in Run mode are the same as we have described in Edit mode: Buses, generators, loads, lines and transformers. However, there are several changes in the representation of the OD in Run mode, compared to Edit mode. Many of the changes depend on how a component "behaves" after interdiction and its "status." This is described next using references to Figure 45:

- Buses: A bus is colored blue if it is "In Service" (e.g., bus "213"). Otherwise, it is colored red if is Interdicted (e.g., bus "216"), or gray if it is Indirectly Interdicted (e.g., bus "211"). Indirectly Interdicted can occur if the bus is located at a substation and that substation is interdicted. In this case, the bus line color remains gray but the bus label (which contains the substation code too) is displayed in red (e.g., buses "209," "210," "211" and "212"). Remark: If a bus is cut off from the rest of the network by the interdiction of a set of surrounding buses, the bus is not considered indirectly interdicted.

- Generators: A generator (which may represent a collection of generators) is colored black if it is In Service (e.g., the generator associated with bus "213," even if its current output is 0 MW). Otherwise, its outline color is red (e.g., the generator associated with bus "216").

- Loads: A load is colored black if its demand is met (e.g., the load associated with bus "213"). Otherwise, if its bus has been Interdicted or Indirectly Interdicted (e.g., bus "216"), the load outline color is red. It may occur that the associated bus is In Service but we need to shed some or all of its load (because of insufficiencient, operational generating and/or transmission capacity in the system). In this case, the load outline color will still black, but the fill color for the triangle will become closer to red as the amount of load shed increases (becoming completely red if 100% of the load is shed; for instance, consider the load associated with bus "214").

- Lines and Transformers: A Line or a Transformer is colored black or green respectively, if it is In Service. Otherwise, its outline color is red if it is Interdicted, or gray it is Indirectly Interdicted.

Figure 45.   First Level of Information in Run Mode. The OD GUI (for a specific Scenario and Period) shows insightful visual information about the system behavior. For example, the red-colored buses and the red-colored transformer have been interdicted. The gray-colored buses, lines and transformers have been indirectly interdicted. Substation "Sub_22" is also interdicted; this is indicated by the red in the labels associated with the bus, which show the combined bus-substation code. The rest of the components are in service. Width and arrow size for lines depends on the power carried by the line, and on the fraction of the line's capacity being used, respectively. Load at bus "108" has been partially shed, whereas at bus "215," no load at all can be met. On the other hand, all the demand is served at bus "206." How the system evolves during for the next restoration period is just one *click* away on the Period list (see Figures 26, 46).

In Run mode, the OD is not editable: The only change that may occur is when the user selects a different Scenario or Period to represent, and the OD changes accordingly.

83

Options, tips and dialogue windows are available to the user as described in the following paragraphs.

b. *Information Displayed*

As in Edit mode, we have information levels of "Permanent," "Tip" and "Advanced" that can be displayed with our OD.

			Met	Shed	Cost	Cum. Shed	Cum. Cost
Scenario	Period	Power	3,951 MW	1,749 MW	$73,367/h		
16	1 (0h - 72h)	Energy	1,137,888 MWh	503,712 MWh	$21,129,696	629,640 MWh	$656,138,520
	2 (72h - 360h)						
	3 (360h - 768h)						
	4 (768h - 1000h)						
x=4332, y=12							

Figure 46.   Scenario Display Area. In this example the user has selected the second restoration Period of Scenario "16." The text boxes show information about the system status for that period of time, such as: Instantaneous power met and shed, total power cost, energy met and shed, and energy cost for the whole period. Cumulative values for energy shed and its cost (from hour "0" to the last hour of the selected period) are also displayed.

The characteristics of the Permanent level are:

- The Scenario Display area (Figure 46) shows the Scenario and Period which the current OD represents. The text boxes on the right-hand side of the display area show information about the system status for that period of time: Instantaneous power met and shed, and total cost. These data are also displayed in terms of energy met and shed and energy cost for the whole period. Cumulative energy shedding and cumulative cost from hour "0" to the last hour of the selected period are also displayed. The user can select a new Scenario and/or Period at any time.

- OD colors and shapes are used to identify the object type, and how the interdiction plan has affected the network's functionality.

- Labels inside or around an object display the following data.   For buses: Bus code and substation where the bus is located (if any).

For generators: Generating output at the bus. For loads: Load met at the bus. For lines and transformers: Power flows (optional).

- Line width: The OD uses thicker line widths to depict lines carrying more power. The default value for the power flow levels is the same as for the capacity levels in Edit mode (see Section E), and it can be modified by the user.

- Line arrow: Arrows show the direction of power flows (no arrow is shown if power flow is zero), and arrow sizes increase as the percentage of utilized line capacity increases.   Default value for the utilized line capacity levels (or "ranges") are: Level 1: [0,50%), Level 2: [50%, 90%), Level 3:[90%, infinity). (Remark: The power flow models used by the optimization module do not allow line overloading, i.e., in practice, Level 3 is [90%, 100%]).

Arrow size for lines is set using the *ArrowMid* property of AddFlow Link object.   The following sizes are used: *Link.ArrowMid* = 15, 30, 45 for Levels 1, 2 and 3, respectively.

The *DrawColor* property is used to set the color of any object contour. In the case of Loads, we also need to paint the interior of the arrow. This is accomplished using the *FillColor* property.

The Bus object painting, which has multiple cases of interdiction, is controlled by the *PaintBus* procedure located at the *DrawDesign* VB module.

```
PaintBus(bus  As  afNode,  lineColor  As  Long,  genColor  As  Long,
loadColor As Long, labelColor As Long)
```

In this procedure, "bus" is a Bus object designed with AddFlow, and the rest of the variables are used to specify the desired color for each of Bus subnodes (BusLine, Generator, Load, Label).

For example, if a substation is interdicted, the associated buses and subsidiary objects in those buses are colored gray, whereas the bus labels are colored red.

The call from our VB code is as follows:

```
PaintBus bus, vbGray, vbGray, vbGray, vbRed
```

Similarly to Edit mode, a second level of information is provided through "Tips" (see Figure 47).

In Run mode, tips display the following information:

- Buses: Code and full name

- Lines and transformers: Code, capacity, In-Service status and power flow.
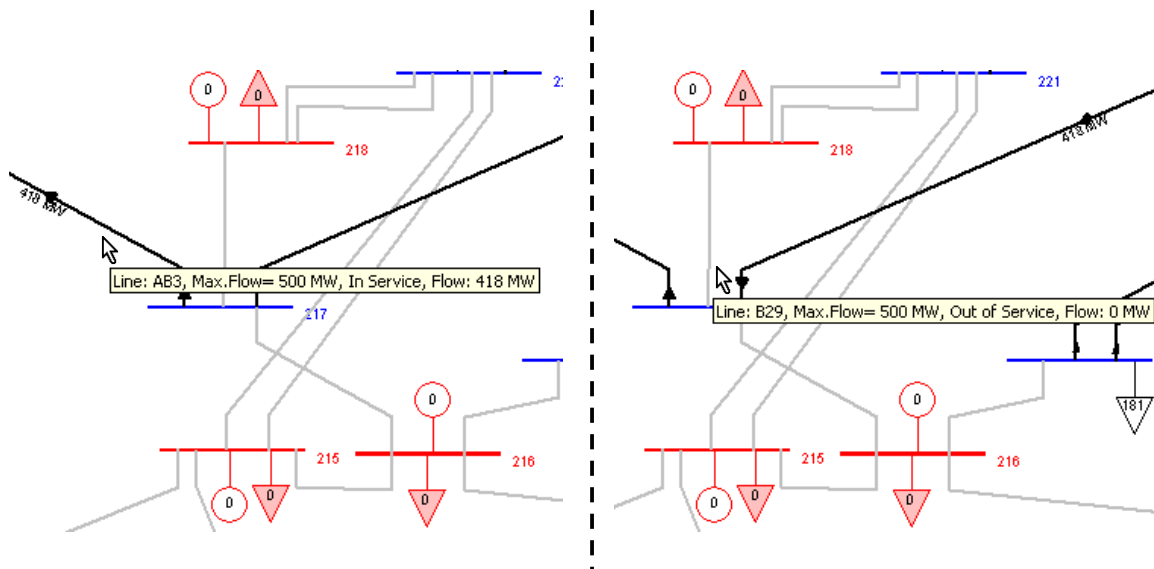

Figure 47.   Tip Level of Information in Run Mode. The tip for the selected line in the left figure shows an "In Service" status for the line, which is carrying 418MW out of a maximum of 500MW. The line in the right-hand figure is "Out of Service." (In fact, its gray color tells us that it has been indirectly interdicted.)

86

Again, dialogue windows constitute the Advanced information level. Access and navigation through dialogue windows is performed much as it is in Edit mode, but the information displayed now incorporates results that were not available in Edit mode. Interdiction and In-Service statuses in dialogue boxes follow the same coloring rules used for the OD.

Dialogues in Run mode do not allow modification of any of the data or results displayed. Therefore, they are not true "Dialogue" windows, although we refer to them as such because of their analogy to Dialogues in Edit mode.

4. Bus Dialogue

The Bus Dialogue is divided into four sections: Title, Generators, Load by Sector, and Lines/Transformers (Figure 48).

The Title section displays the bus code and name, along with the In-Service status. For example, bus "107" is In-Service during the selected period, and therefore is displayed in blue, whereas bus "215's" status is Not-In-Service, which is represented in red if the bus has been interdicted, or in gray if it has been indirectly interdicted.

The Generator section, displayed in tabular form, provides information about every generating unit at the bus and total for the whole bus. In keeping with the OD coloring rules, Interdicted Generators are displayed in red, or in gray if the interdiction is indirect (as in the case of generators at bus "215").

The Load by Sector section displays the power met and shed at the bus for every sector, where shedding amounts appear in red.

The Lines/Transformers section displays line status, line capacity and power flow on the line, along with the bus name and code at the other end of the line. The line code and the transformer code are displayed in an appropriate color according to the In-Service status and their Interdiction status.

As in Edit mode, the user can navigate to the dialogue window of any of the listed lines or transformers by *double-clicking* on their codes.
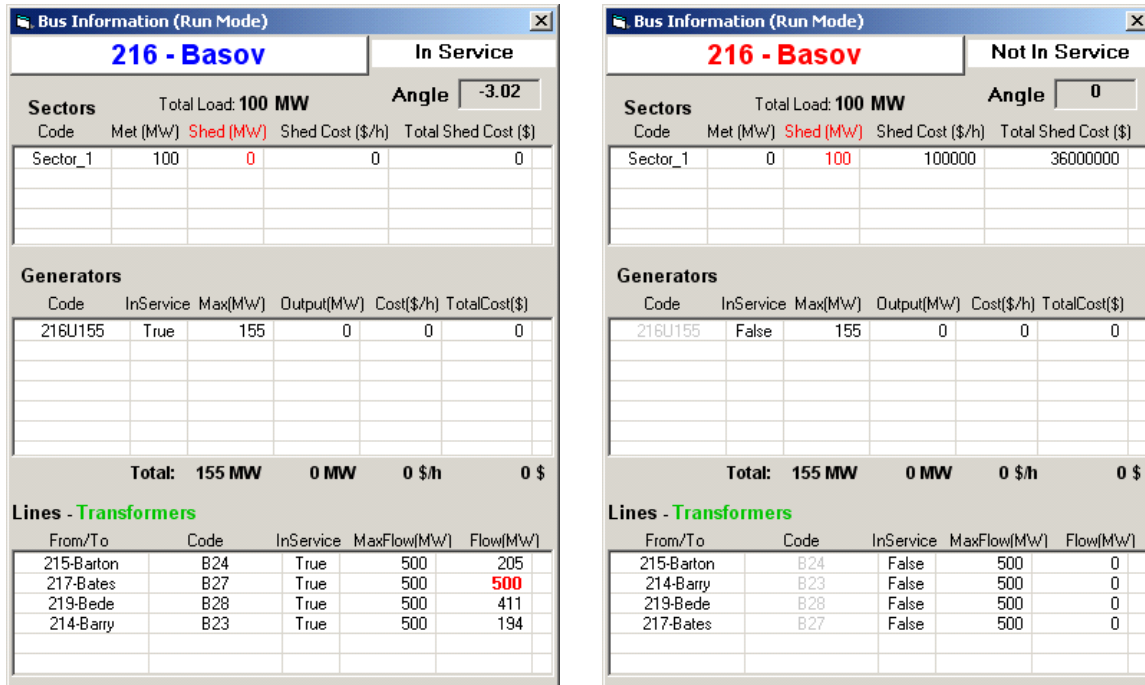
Figure 48.   Bus Dialogue in Run Mode. The two panels show the same bus ("216-Basov") under two different statuses. On the left dialogue, the bus is "In Service." In fact, we can see that all load at the bus has been met ("shed" is zero for the only sector of the problem). The only generator connected to the bus is not producing any power, but this is not due to any interdiction, because its code is colored in black. The bottom part of the window shows the lines and transformers connected to the bus. One line ("217-Bates") is loaded at its maximum capacity (500MW). In the right-hand dialogue, the bus has been interdicted, which causes an indirect interdiction of all associated generators and lines, as well as a complete blackout for the load connected to the bus.

5.   Line/Transformer Dialogue

The Line/Transformer Dialogue Window (Figure 49) is divided into the following areas: Title, Interdiction, Capacity and Buses.

The Title section displays the line or transformer code. The code is displayed in black for a line or in green for a transformer. However, depending on the In-Service and Interdiction statuses of the line, these colors can change to red or gray.
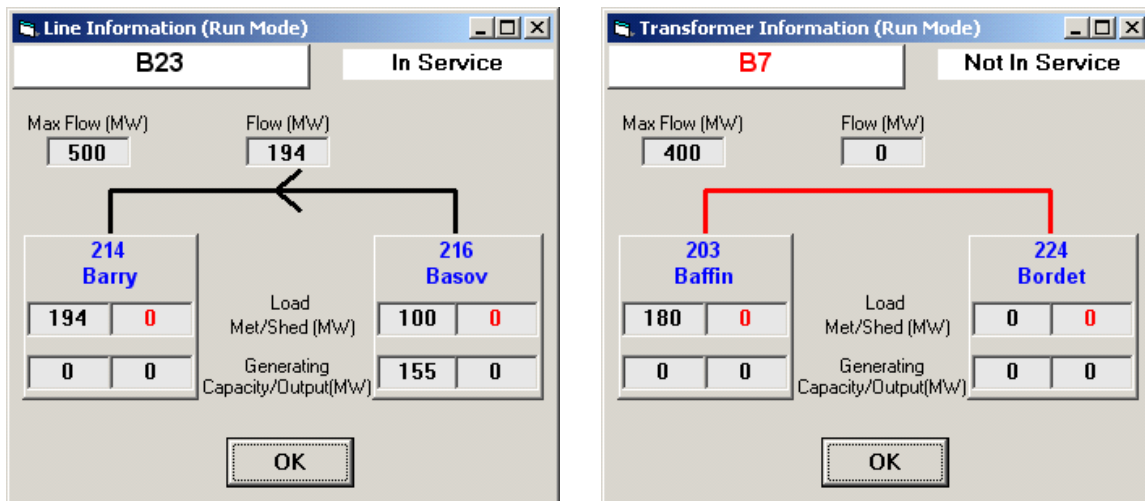
Figure 49.   Line/Transformer Dialogue in Run Mode.   The example on the left dialogue shows information about line "B23," which is in service, and carries 194MW. The line connects buses "216-Basov" and "214-Barry," neither of which has been interdicted since both are colored in blue.   In the right-hand dialogue, the dialogue window for an interdicted transformer (notice the red color) is shown.

The Capacity section (Max Flow and Flow text boxes) shows the line capacity along with the actual power flow on the line.

The Bus section shows the two buses associated with the line or transformer, connected by black or green lines, respectively.   The line color follows the code color in the Title section.   The line includes an arrow showing the direction of the flow between both buses.   Additional load and generation information for each bus is displayed.

The user can navigate to the dialogue window of any of the two buses by double-clicking on the code-name area for the desired bus.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

A. CONCLUSIONS

This thesis has developed a graphical user interface (GUI) to represent electric power grids subject to interdiction (attack) by terrorists. The work has improved an instrumental component of the VEGA (Vulnerability of Electrical Power Grids Analysis) GUI that represents the problem graphically using One-line Diagrams (ODs). Conforming to Microsoft Windows standards, the new OD GUI incorporates advanced graphical features, which help the user visualize and understand the effects of interdiction. Examples of improvements over the previous OD GUI in VEGA 1.0 are:

- Graphical representation of "dynamic system restoration," i.e., changing system behavior (power flows, shed load, etc.), over time, as interdicted components are repaired.

- Helpful dialogue windows that allow viewing extended data and results associated with individual network components.

- Graphical representation of power flows, using arrows and values in the OD.

- Printing capabilities.

- More flexibility: allows object movement, zooming, rotating objects, etc., which in turn causes an overall nicer visual effect than that of VEGA 1.0.

- Easier extensibility: the new OD GUI has been built using a technology that will greatly simplify the programming necessary to extend of the application and incorporate new features.

The enhanced OD GUI has been incorporated into the new version of the system, VEGA 2.0.

One of the most important VB limitations we have overcome is the poor toolset available to create the complex graphical structures required in VEGA. A key piece in solving this problem is the ActiveX control called AddFlow, purchased from a vendor. This control is the basis for all the OD graphical objects constructed in the new GUI.

The OD objects (and sub-objects within objects) created from AddFlow native objects, Node and Link, exhibit great flexibility in terms of visibility and transparency, drawing options, anchoring and movement, mouse-controlled events, printing, etc.

The enhanced OD GUI developed in this thesis provides insightful visualization of power-flow disruption caused by terrorist attacks, letting the user compare different attack plans for multiple scenarios, or analyze system restoration dynamics for a particular scenario, or simply stop at a given scenario and point in time in order to analyze the whole system in detail. To help in this last analysis, dialogue windows have been created and customized in order to show selected information for the system components. In addition, dialogue windows allow quick access to data and results, by clicking on the desired system component in the OD.

Overall, we believe the OD GUI improvements described in this thesis represent an important step forward in the development of the VEGA decision-support system.

B.   RECOMMENDATIONS FOR FUTURE WORK

A number of potentially useful features remain to be implemented in the OD GUI, and some known difficulties in using it have not been fully addressed. Some areas in which further work is needed include:

- More effective "case-loading:" In very large cases, most of the buses will naturally be set to "non-visible" by the user. However, even in this instance, all the existing components are loaded as objects in the OD. Currently, loading a real-world case with about 5,000 buses and lines can take more than

one minute in a 3MHz personal computer with 1Mb of RAM. A suggested solution to avoid this delay consists of not to overburden the OD display by creating objects that are meant to be invisible. However, the current design of Dialogue Windows requires all objects to be created in order to display the object information in those windows. The implementation design could be partially modified, for example, by creating these invisible objects temporarily, and only when needed.

- More than one OD per case: The user may want to save and retrieve more than one OD view for the same case (for example, with a different number of visible buses, or using a different coordinate system). This capability does not currently exist, but to add it would require redesigning the portion of the DB that stores the OD configuration.

- Advanced Print and Report Capabilities: The OD GUI has limited printing capabilities. It would be useful to be able to print additional information such as the text boxes that appear in scenario display area (Chapter IV Section B3), or inside the dialogue windows. These and other advanced printing features can be implemented using the existing properties of *PrnFlow*. Similarly, it would be desirable to incorporate formal reports with information on such things as data and results. Reports could be incorporated using Crystal Reports [Crystal Decisions 2003] or other reporting tools.

- "Equivalent" graphics: "Equivalencing" is a technique used in electrical engineering to reduce a large electrical power system into a smaller system with similar properties, basically by grouping components. Independent of whether the case we represent in our OD GUI is in itself a case that has already undergone "equivalencing," we need to be able apply some type of "result equivalencing" when some of the buses are invisible. Their information (power flows, loads and generation) should be picked up by some of the visible buses. In other words, the hidden information should be displayed as results associated with the visible buses, in some aggregated fashion. Addressing this

problem will require the definition of criteria for aggregating and displaying the information, as well as the creation of new objects and procedures.

- Help files: In order to produce a more user-friendly program, the creation of an online Help (whether this is contextual or provided as a separate menu) is necessary.

- Conclude the implementation of the Options window.

APPENDIX. ADDFLOW USEFUL FEATURES

The following tables display particular features (mostly properties) for Nodes, Links, LinkPoints and common collections, respectively.

Name	Type	Description
AdjustDst	P	Determines whether or not the destination point of Link objects can be adjusted by the user
AdjustOrg	P	Determines whether or not the origin point of Link objects can be adjusted by the user
Click	E	Triggered when the user presses and then releases a mouse button over the control
Copy	M	Copies a diagram (or a part of it) into the clipboard
DblClick	E	Triggered when the user presses and releases a mouse button twice over the control
DisplayHandles	P	Determines whether the handles used for selection are displayed or not
DrawColor	P	Returns/sets the default pen color used to draw objects (Node or Link)
DrawStyle	P	Returns/sets the default pen style used to draw objects (Node or Link)
DrawWidth	P	Returns/sets the default pen width used to draw objects (Node or Link)
Enable	P	Returns/sets a value that determines whether an object is enabled or not
ExportPicture	M	Exports a diagram (or a part of it) as an enhanced metafile picture
FillColor	P	Returns/sets the default color used to fill Node objects
ForeColor	P	Returns/sets the default foreground color used to display text
GotFocus	E	Triggered when the object receives the focus. This event is automatically added by VB. (In other environments, use afGotFocus.)
Height	P	Returns/sets the height of an object
Hidden	P	Determines whether objects (Node or Link) are by default visible or hidden
Left	P	Returns/sets the distance between the internal left edge of an object and the left edge of its container
LinkStyle	P	Returns/sets the default style (polyline, bezier) used to draw Link objects

Name	Type	Description
MouseDown	E	Triggered when the user presses the mouse button while an object has the focus.
MouseMove	E	Triggered when the user moves the mouse
MouseUp	E	Triggered when the user releases the mouse button while an object has the focus
Nodes	P	Returns a reference to the collection of all Node objects of the diagram
OrientedText	P	Determines whether the text of a link can be drawn in the same direction as the link itself
ProportionalBars	P	Determines whether the scrollbar thumbs should be proportional to the size of the visible area
Repaint	P	Determines whether repainting the control is allowed or not
Rigid	P	Determines whether Link objects are by default rigid or not
ScrollBars	P	Allows adding scrollbars for the control
SelectedLink	P	Returns/sets a value which determines if a Link object is selected
SelectedNode	P	Returns/sets a value which determines if a Node object is selected
ShowToolTip	P	Determines whether Node and Link tooltips should be displayed or not
SizeArrowMid	P	Returns/sets the default Link object middle segment arrow size
SizeArrowOrg	P	Returns/sets the default Link object destination arrow size
Tag	P	Stores a tag (customized extra text) typically used for recognizing some aspect of the object within the program
Top	P	Returns/sets the distance between the internal top edge of an object and the top edge of its container
Transparent	P	Determines whether a Node object is transparent or not
Visible	P	Returns/sets a value that determines whether an object is visible or hidden
Undo	M	Undoes, if possible, the last action
Width	P	Returns/sets the width of an object
XZoom	P	Returns/sets the horizontal zooming factor
YZoom	P	Returns/sets the vertical zooming factor

Table 2.   List of AddFlow properties (P), events (E) and methods (M).

Name	Description
Alignment	Sets or returns the alignment style of text in the Node
DrawColor	Returns/sets the pen color used to draw the Node
DrawStyle	Returns/sets the pen style used to draw the Node
DrawWidth	Returns/sets the pen width used to draw the Node
EditMode	Determines whether the text of the Node can be edited or not
FillColor	Returns/sets the color used to fill the Node
Font	Returns/sets the font used to display the Node text
ForeColor	Returns/sets the foreground color used to display the Node text
Height	Returns/sets the height of the bounding rectangle of the Node
Hidden	Determines whether the Node is visible or hidden
Index	Returns the index of the Node in the Nodes collectio.
InLinks	Returns a reference to the collection of incoming Links for the Node
Left	Returns/sets the left position of the bounding rectangle of the Node
Links	Returns a reference to the collection of all incoming and outgoing Links for the Node
Marked	Returns/sets a flag associated with the Node
OutLinks	Returns a reference to the collection of outgoing Links for the Node
Selectable	Determines if the Node can be selected by the user or not
Selected	Determines if the Node is currently selected or not
Sizeable	Determines whether the Node can be resized using the mouse or not
Tag	Returns/sets the tag associated with the Node
Text	Returns/sets the text associated with the Node
ToolTip	Returns/sets the Node tooltip
Top	Returns/sets the top position of the bounding rectangle of the Node
Transparent	Determines whether the Node is transparent or not
UserData	Returns/sets a customized numeric datum associated with the Node
Width	Returns/sets the width of the bounding rectangle of the Node
xMoveable	Determines whether a Node can be moved horizontally or not

Name	Description
yMoveable	Determines whether a Node can be moved vertically or not
ZOrder	Places a Node at the front or back of the Z-order list. Remark: ZOrder is a mehtod, not a property.
ZOrderIndex	Returns/sets the position of the Node in the Z-order list

Table 3.   List of Node properties.

Property	Description
DrawColor	Returns/sets the pen color used to draw the Link.
DrawStyle	Returns/sets the pen style used to draw the Link
DrawWidth	Returns/sets the pen width used to draw the Link
ExtraPoints	Returns a reference to the collection of the Link points
Dst	Returns/sets the reference of the destination Node of the Link
ForeColor	Returns/sets the foreground color used to display the Link text
Hidden	Determines whether the Link is visible or hidden
LinkStyle	Returns/sets the style (polyline, bezier) used to draw the Link
Marked	Returns/sets a flag associated with the Link
Org	Returns/sets the reference of the origin Node object of the Link
Reverse	Reverses the origin and the destination of a Link
Rigid	Determines if the Link is rigid or not
Selectable	Determines if the Link can be selected by the user or not
Selected	Determines if the Link is selected or not
ShowJump	Returns/sets a value which determines whether jumps are displayed at the intersection of the Link with other Links or straight crosses are used
Tag	Returns/sets the tag associated with the Link
Text	Returns/sets the text associated with the Link
ToolTip	Returns/sets the Link displayed tooltip
UserData	Returns/sets a customized numeric data associated with the Link

Property	Description
ZOrder	Places a Link at the front or back of the Z-order list. Remark: ZOrder is a method, not a property

Table 4.   List of Link properties.

Property	Description
x	Horizontal coordinate of a point
y	Vertical coordinate of a point

Table 5.   List of LinkPoint properties.

Name	Type	Description
Add	M	Adds an item to a Collection.
Clear	M	Erases all the items in a Collection.
Count	P	Returns the number of item objects in a Collection.
Item	P	Returns the reference to an item object of a Collection.
Remove	M	Removes an item from a Collection.

Table 6.   List of Collection properties (P) and methods (M).

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Balena, F. (1999). *Programming Microsoft Visual Basic 6.0*, 1st edition, Microsoft Press, Redmond, Washington.

Brooke, A., Kendrick, D., Meeraus, A. and Raman R. (1998). *GAMS: A User's Guide*. GAMS Development Corporation.

Chan, S. (1990). "Interactive Graphics Interface for Power System Network Analysis," *IEEE Computer Applications in Power*, January, pp. 34-38.

Crystal Decisions (2003). http://www.crystaldecisions.com/ (accessed 15 Dec 2003)

Department of Defense (2002). *Joint Technical Architecture*, Version 4.0, pp. 71-74.

EIA (Energy Information Administration) (2003). http://www.eia.doe.gov/cneaf/electricity/page/glossary.html (accessed 15 Dec 2003)

Elec-Saver (2003). http://www.elec-saver.com/e-defs.htm#a) (accessed 15 Dec 2003)

GAMS (2003). http://www.gams.com (accessed 15 Dec 2003)

Lassalle Technologies (2003). http://www.lassalle.com/ (accessed 15 Dec 2003)

Microsoft (1998). Microsoft Visual Basic 6.0 Programmer's Guide, Microsoft Press, Redmond, Washington

Microsoft (2003-I). Microsoft Visual Basic 6.0, http://msdn.microsoft.com/vbasic/ (accessed 15 Dec 2003)

Microsoft (2003-II). http://www.microsoft.com (accessed 15 Dec 2003)

Overbye, T., Sauer, P., Marzinzik, C. and Gross, G. (1995). "A User-Friendly Simulation Program for Teaching Power System Operations," *IEEE Transactions on Power Systems*, 10-4, pp. 1725-1733.

PowerWorld (2003). http://www.powerworld.com/ferc/ferc715.html (accessed 15 Dec 2003)

Salmeron, J., Wood, K. and Baldick, R. (2003-I). "Optimizing Electric Grid Design under Asymmetric Threat," NPS-OR-03-002 Technical Report, Operations Research Department, Naval Postgraduate School, Monterey, California.

Salmeron, J., Wood, K. and Baldick, R. (2003-II). "Analysis of Electric Grid Security under Terrorist Threat," *IEEE Transactions on Power Systems*, to appear.

University of Washington (2003). http://www.ee.washington.edu/research/pstca/ (accessed 15 Dec 2003)

VEGA Project (2003). http://www.nps.navy.mil/or/research/VEGA/vega.htm (accessed 15 Dec 2003)

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Javier Salmeron, Code OR/Sa
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

4. Kevin Wood, Code OR/Wd
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

5. Ted Lewis, Code CS/Lt
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

6. Paul Stockton
   Center of Homeland Security
   Naval Postgraduate School
   Monterey, California

7. Ross Baldick
   Department of Electrical Engineering
   University of Texas at Austin
   Austin, TX, 78712-1084

8. Major Dimitrios Stathakos
   Hellenic Army General Staff, DEPLH
   Stratopedo Papagou, Xolargos
   Athens, Greece